Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study

Heesch, U. van; Avgeriou, P.; Tang, A.

Published in:
Journal of Systems and Software

DOI:
10.1016/j.jss.2013.01.057

The Journal of Systems and Software 86 (2013) 1545–1565

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study

U. van Heesch [a,*], P. Avgeriou [a], A. Tang [b]

[a] University of Groningen, Groningen, The Netherlands
[b] Swinburne University of Technology, Melbourne, Australia

ARTICLE INFO

Article history:
Received 6 July 2012
Received in revised form 20 January 2013
Accepted 24 January 2013
Available online 17 February 2013

Keywords:
Software architecture
Architecture decisions
Viewpoints
ISO/IEC/IEEE 42010
Design reasoning
Case study

ABSTRACT

Software architecture design is challenging, especially for junior software designers. Lacking practice and experience, junior designers need process support in order to make rational architecture decisions. In this paper, we present the results of a comparative multiple-case study conducted to find out if decision viewpoints from van Heesch et al. (2012, in press) can provide such a support. The case study was conducted with four teams of software engineering students working in industrial software projects. Two of the four teams were instructed to document their decisions using decision viewpoints; the other two teams were not instructed to do so. We observed the students for a period of seven weeks by conducting weekly focus groups and by analyzing their work artifacts and minutes. Our findings suggest that junior designers who use decision viewpoints are more systematic in exploring and evaluating solution options. However, the decision viewpoints did not help them in managing requirements and complexity.

1. Motivation and background

Software architecture design is a demanding task which requires designers to find optimal solutions within a specified timeframe for often vaguely defined requirements, while managing risks, regarding constraints, and taking business drivers into account. There is a steep learning curve to becoming a good architect: junior software designers usually have to endure extensive periods of learning, going through numerous painful trial and error attempts when making architecture decisions.

Before becoming software architects, junior software designers need to develop (a) a certain body of knowledge, and (b) the cognitive skills for systematically reasoning about architecture decisions. These two factors are important for making rational decisions. A rational decision is a decision based on the application of reason. A rational decision deliberates the benefits and drawbacks of the available design options, while taking requirements and other project constraints into account.

Junior software designers need guidance to handle the inherent complexity of rational decision making, especially with software architecture issues. Explicitly modeling architecture decisions during the design process may provide such a guidance. Although being widely overlooked in software engineering education, the treatment of architecture decisions as first class entities has gained increasing attention in the software architecture research field and also in industrial practice. In recent years, many authors have stressed the importance of thoroughly documenting architecture decisions in software projects (e.g. Tyree and Akerman, 2005; Kruchten, 2004; Tang et al., 2010; Jansen and Bosch, 2005).

Initially, the main perceived benefit of documenting architecture decisions was to share a common understanding of the developed architecture between stakeholders like architects, developers, and customers (Tyree and Akerman, 2005; van der Ven et al., 2006), primarily to ease change, maintenance, and evolution of the architectural design. Kruchten later stressed that the modeling of (potential) decisions, particularly their dependencies and interrelations, can also support the architect when reasoning about the decisions (Kruchten, 2004). In our previous work, we developed a documentation framework for architecture decisions that addresses many stakeholder concerns in architecture decisions (van Heesch et al., 2012, in press). Using the conventions of the international architecture description standard ISO/IEC/IEEE 42010 (ISO, 2011), the framework provides five viewpoints for architecture decisions, each of which being designed to address different decision-related concerns:

Decision forces viewpoint: It makes explicit the relationships between architectural decisions and the forces that influenced the architect when making the decisions out of multiple

* Corresponding author. Tel.: +49 151 172 830 24.
E-mail addresses: uwe@vanheesch.net, uwe.van.heesch@capgemini.com (U. van Heesch), paris@cs.rug.nl (P. Avgeriou), atang@swin.edu.au (A. Tang).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.01.057

alternatives. In this context, a *force* is "any aspect of an architectural problem arising in the system or its environment (operational, development, business, organizational, political, economic, legal, regulatory, ecological, social, etc.), to be considered when choosing among the available decision alternatives" (van Heesch et al., in press).

Decision relationship viewpoint: It makes explicit the relationships between architecture decisions (e.g. depends on, caused by, or is alternative to).

Decision chronology viewpoint: It shows the evolution of architecture decisions over time.

Decision stakeholder involvement viewpoint: It describes the roles of specific stakeholders in the decision-making process, capturing which stakeholders proposed, confirmed, or validated specific decisions.

Decision detail viewpoint: It gives detailed information about single architecture decisions, including a comprehensive description of the chosen architectural solution and the rationale for choosing this solution.

Building on the idea that modeling decisions supports the design process of the architecture, we conjecture that, besides being a useful tool for professional architects, decision viewpoints can particularly guide junior software designers, helping them to make rational decisions. The question is in which areas of software architecting can decision viewpoints help to guide designers.

In this paper, we report on a comparative multiple-case study conducted with four groups of senior software engineering students (near graduation), to find out if modeling design decisions supports them in following a rational design process. We selected three decision viewpoints from our framework that particularly frame concerns related to decision making support: the *decision detail viewpoint*, the *decision relationship viewpoint* (both from van Heesch et al., 2012), and the *decision forces viewpoint* (defined in van Heesch et al., in press). The results show that particularly the decision forces viewpoint and the decision relationship viewpoint supported the students to systematically identify and evaluate multiple decision alternatives for the design problems.

The rest of this paper is organized as follows. Section 2 presents the design of the study including the study goal, research conjecture and response variables. Section 3 reports on the data analysis and interpretation. In Section 4, we discuss potential threats to the validity of our findings. We discuss related work in Section 5 and present our conclusions in Section 6.

2. Study design

The goal of the study is to explore if junior designers, who document decision views according to the decision viewpoint framework presented in van Heesch et al. (2012), use more of a rational design approach than junior designers with an ad hoc approach. The study is comparative in nature. We try to evaluate the influence of decision view creation on the use of a rational design process.

In cases, where the cause–effect relationship between a specific treatment (in this case the decision view creation) and an outcome (a rational design process) is to be observed, formal experiments can be taken into consideration as empirical method. However, experiments require careful control of the so-called independent variables, which represent potential factors that influence the outcome of the study, to ensure that outcomes are results of the applied treatments. To achieve this control, experiments are usually conducted in a laboratory environment (Wohlin et al., 2000), in which confounding factors can be eliminated, and independent variables can be carefully controlled at pre-determined levels.

Studying the impact of decision view creation (the treatment) on the design process (the outcome) does not allow for this level of control. Apart from the fact that the design process takes multiple weeks, the impact could be wide-ranging, covering multiple aspects of the students' behavior and the project itself. Reducing the measurement to a set of predefined variables would be inappropriate in this case. Additionally, providing a fictional case with artificial requirements and virtual customers would have introduced a threat to validity, as the design of the fictional case could influence the outcome of the study. If conducted as an experiment, the study and its results could be considered as unrealistic and not transferable to industrial practice.

Case studies, on the other hand, are well suited for studying objects of study that are hard to study in isolation (Runeson and Höst, 2009). They provide a deeper understanding of the situation under study than experiments do. Case studies are suitable for understanding real-life events (Yin, 2009). Yin points out that case studies are preferable over experiments in cases, in which control of behavioral events is not possible or not required (Yin, 2009). Yet, single case studies are not suitable for doing comparisons, because they are lacking a reference that can be used as a basis for the comparison. This problem has been addressed by Kitchenham et al., who provide guidelines for planning and conducting case studies for the evaluation of software engineering methods (Kitchenham et al., 1995). The guidelines combine the advantages of case study research and formal experiments, and they are well established in the empirical research community (Sjoberg et al., 2007; Easterbrook et al., 2008; Host and Runeson, 2007). In order to allow for the comparison of two software engineering methods, three types of case study arrangements are distinguished:

- Conducting a single case study in one software project and comparing the results against a company baseline, for which empirical data is readily available.
- Conducting a single case study in one software project, using the decision viewpoints for a subset of decisions, while using a different method for the other architecture decisions that need to be made.
- Conducting multiple case studies, in which decision viewpoints are applied to a set of software projects and compared to the results of another set of software projects (the so-called sister projects).

The first type of arrangement is not feasible, because there is no company baseline against which the students' design activities could be compared. The second arrangement was ruled out, because it would not make sense to ask the students to use the decision framework for some decisions, while forbidding its use for other decisions. In such a scenario, it would have been impossible to avoid maturation effects (Wohlin et al., 2000); e.g. the students would have become more familiar with the problem space or already have a more concrete idea of the overall architecture when using the other method. Therefore, we decided to apply the third type in our study: conducting multiple case studies, in which half of the project teams apply our decision documentation approach, while the other half follows an ad hoc way of designing and documenting. Recently, several other comparative studies have successfully used case studies to evaluate software engineering methods (Nagappan et al., 2008; Jiang et al., 2008; Serral et al., 2010).

We use general guidelines for conducting and reporting on case studies defined by Runeson and Höst (2009) and synthesize them with the comparison method suggested by Kitchenham et al.; this

implies the following additions to the case study method:

- The case study context needs to describe the baseline (gathered from the sister project), against which the impact of the decision viewpoints is compared.
- An evaluation conjecture needs to be defined. Kitchenham et al. use the term *hypothesis*, but we decided to use *conjecture* instead to clearly differentiate from formal experiments.
- Response variables and data collection methods must be defined. The variables correspond to the criteria used to measure the impact of decision viewpoints on the design activities.
- Case variables describing the characteristics of the projects and the development team need to be defined. These variables correspond to independent variables in experimentation, with the difference that they cannot be controlled, but only described in case studies.

Additionally, we used guidelines and suggestions for planning and reporting on case studies (Verner et al., 2009; Brereton et al., 2008).

2.1. Context, research goal and conjecture

The study was conducted in the context of the so-called *Software Factories* (SOFA), a lecturing module at the Fontys University of Applied Sciences in Venlo, in the Netherlands. In this module, groups of students work in software projects for external, industrial customers. The students work on their own responsibility, as if they had founded their own software company. Thus, they are responsible for communicating with the customer, identifying and assigning tasks, and solving all kinds of problems that occur during software projects. A lecturer, who observes their process, accompanies each of the project teams.

After a project runtime of 20 weeks, each of the students is individually assessed by two lecturers. They are graded for their individual performance, the quality of the end product, and the satisfaction of the external customer. None of the researchers was involved in the lecturing module, nor did they have any influence on the grading of the students. The data collected on behalf of the study was not provided to the lecturers, and any publication of the study results takes place after the students received their grades.

The university defines the following project constraints and facilities for the students:

- The students are strongly advised to follow the agile software development process Scrum (Schwaber and Beedle, 2002). One of the students takes the role of the scrum master. Additionally, they have to write minutes for every team meeting.
- Mandatory use of the project management system Trac (Edgewall Software, 2012), which provides a Wiki, reporting facilities, and a web interface for the version control system Subversion (Tigris.org., 2012). Subversion usage is mandatory to store all work artifacts created during the project including source code and configuration files, all project documentation, design artifacts, minutes, and Scrum-specific artifacts like user stories, for instance.
- The students have to work on site at the university for at least three complete working days (8 h) per week. Therefore, each of the project teams is provided with its own office, whiteboards, and a projector.

Using the goal definition technique suggested in the *goal, question, metric* approach (Basili et al., 1994), the overall goal of the study is to:

Analyze the software development processes of senior software engineering students working in groups of 4–6 people **for the purpose of** evaluating the supportive effect of architecture decision view creation on their decision making process **with respect to** reasoning best practices identified in our previous studies (van Heesch and Avgeriou, 2011) **from the point of view of** external empirical researchers **in the context of** the software factories course at the Fontys University of Applied Sciences in Venlo, the Netherlands.

A cost-benefit analysis, in which the advantages of using decision-viewpoints are contrasted against the effort for creating decision views, is not a goal of this study, as such an analysis was part of an earlier study on decision viewpoints with professionals (van Heesch et al., 2012). Based on our previous experience with students who created decision views in their software projects, we derive the following research conjecture from the study goal:

RC: We conjecture that student groups (decision view group) who work in a software project follow a more rational design process if they iteratively create and refine architecture decision views, compared to student groups (comparison group) who follow an ad hoc approach.

The rationality of the design process, as referred to in this conjecture, is evaluated by using a set of eleven response variables, defined in the following subsection.

2.2. Response variables

In this section, we present the response variables used to determine the rationality of the students' design process. Particularly, the variables are used to find out which reasoning practices the students follow during the design. We use reasoning best practices, identified in our previous work with professional software architects from the industry (van Heesch and Avgeriou, 2011). Each variable poses a question that the study is trying to answer.

Code	**Resp1**
Design activity	Identification of architecture significant requirements (ASRs)
Description	How do the students elicit requirements in general and how do they identify requirements that need to be considered when making architecture decisions?
Code	**Resp2**
Design activity	Requirements negotiation
Description	How do the students negotiate requirements with the project stakeholders? Requirements could be negotiated, for instance, if they unnecessarily impede the project progress, if they are unrealistically challenging, or if they are not state-of-the-art.
Code	**Resp3**
Design activity	Prioritization of requirements
Description	How do the students prioritize requirements when identifying architectural approaches? Attention is drawn in particular to the order and effort put in finding candidate solutions to address specific requirements.
Code	**Resp4**
Design activity	Documentation of requirements
Description	How do the students document requirements? Particular attention is payed to the S.M.A.R.T. characteristics (Mannion and Keepence, 1995) *specific*, *measurable*, *attainable*, *realizable*, and *traceable*
Code	**Resp5**
Design activity	Discovery of design options
Description	How do the students identify design options to address architectural problems?

Code	**Resp6**
Design activity	Balancing advantages and disadvantages of design options
Description	How do the students consider the advantages and the disadvantages when selecting a solution out of multiple design options during architectural evaluation (with architectural evaluation, we refer to the process of choosing out of multiple design options, as defined in Hofmeister et al., 2007)?

Code	**Resp7**
Design activity	Discussion of multiple design options in combination
Description	Do the students discuss multiple architectural approaches in combination?

Code	**Resp8**
Design activity	Avoidance of unnecessary complexity
Description	Do the students proactively take measures to avoid unnecessary complexity in the architectural design?

Code	**Resp9**
Design activity	Validation of design options against the ASRs
Description	How do the students validate design options against the architecture significant requirements during architectural evaluation? This variable includes the making of compromises in cases where a design option has conflicting influences on multiple ASRs.

Code	**Resp10**
Design activity	Prototyping of design options
Description	Do the students build prototypes, and if so, what are they used for?

Code	**Resp11**
Design activity	Evaluation of the architecture as a whole
Description	How do the students evaluate their designed architectures?

2.3. Case variables

In the following, we define case variables concerning the software projects and the participants of the study:

Code	**CaseVar1**
Name	Study group
Description	This variable describes, if the students in a project document decision views during the design (decision view group), or not (comparison group).
Scale type	Nominal
Unit	n.a.
Range	'decision view group','comparison group'

Code	**CaseVar2**
Name	Programming experience
Description	The programming experience is one of the measures used to describe the software engineering experience of the subjects.The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding theirprogramming experience. We take both, industrial programming experience and academic experience into account.
Scale type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1–3 years, 3–7 years, >8 years

Code	**CaseVar3**
Name	Software design experience
Description	Similarly to the programming experience, the software design experience of the students could have an effect on the outcome of the study. The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding their design experience. As for the programming experience, we take both, industrial experience and academic experience into account.
Scale type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1–3 years, 3–7 years, >8 years

Code	**CaseVar4**
Name	Industrial experience
Description	The industrial experience is expressed as the number of years, the students have worked as a software engineer in the industry (i.e. not in an academic context); for instance in a side job, or prior to the study. The effect of this variable on the outcome of the study is reduced by the fact that the students in the two study groups are balanced regarding their industrial experience.
Scale type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1–3 years, 3–7 years, >8 years

Code	**CaseVar5**
Name	Project domain
Description	The domain of the projects could have an influence on the design activities, as some domains like healthcare or embedded systems require designers to think more carefully about safety critical decisions.
Scale type	Nominal
Unit	n.a.
Range	Possible values: Avionics, Command and Control, Embedded Systems, E-Commerce, Enterprise Computing, Finance, Healthcare, Realtime, Manufacturing, Software Engineering, Scientific, Simulation, Telecommunication, Transportation, Utilities, Marketing, Logistics, Web Applications, Others

Code	**CaseVar6**
Name	Difficulty of the project
Description	The difficulty of the SOFA projects could theoretically influence the design activities followed by the students. Difficulty, in this context, refers to the difficulty of the problem. Judging the difficulty of a project based on objective metrics is challenging and vulnerable. Therefore, we decided to estimate the difficulty of the projects by asking the four lecturers, who supervise and gradethe SOFA projects, to rate the difficulty of each SOFA project. The lecturers were asked to take into consideration the project goals, technologies that would have to be used, as well as the students' previous knowledge in the project domains. Each of the lecturers was knowledgeable about two projects, because they acted as a supervisor for one project, and as assessor for another project. In addition, the researchers judged the difficulty of the projects, using the same criteria as the lecturers. The difficulty of the projects was then calculated by taking the median value of the three given ratings (supervising lecturer, assessing lecturer, and researchers)
Scale type	Ordinal
Unit	n.a.
Range	5-point Likert scale: 1 for *very simple* to 5 for *very difficult*

Code	**CaseVar7**
Name	Experience in the project domain
Description	This variable refers to the experience of the students in the domain of the respective SOFA project (variable CaseVar5). The domain experience could for instance have an influence on the effort in or intensity of exploring the problem and solution spaces.
Scale type	Ordinal
Unit	Years
Range	4 classes: 0 years, 1–3 years, 3–7 years, >8 years

Code	**CaseVar8**
Name	Scrum process followed
Description	All four project teams were advised to follow the Scrum development method. This variable qualitatively describes the degree, to which the project teams actually used Scrum.
Scale type	Nominal
Unit	n/a
Range	Open

2.4. Cases, objects and subjects description

In this section, we explain the four cases. Additionally, important characteristics of the subjects, the sampling procedure, and the object under study will be further elaborated.

2.4.1. Cases and objects

In total, we observed four different software projects run as part of the Software Factory module. In the following, each project will be briefly described. The customer of one of the projects asked for anonymity. As a consequence, we decided to use pseudonyms for all projects.

Project alpha: This project is a brown-field, dealing with a legacy text system, which is used to dynamically generate multiple types of documents based on templates and information stored in a database. Using the templates, data can be composed in multiple ways before being assembled into document formats like PDF, for instance. The Bavarian Department of Justice is one of the prominent users of the system. The primary task of this project team is an architectural re-design to a service oriented architecture (SOA). The customer of the project, a medium-sized German software company, wants to migrate all business-critical services to SOA in the long term.

Project beta: The customer of this project is a Dutch personnel service for chefs (cooks),[1] specialized on temporary arrangements like catering, cook workshops, or interim executive chefs. The primary task of the SOFA project team is the development of a software platform for online personnel services in the gastronomy business, where freelancing cooks can register and apply for jobs, which are posted to the site by restaurant owners, for instance. The project is a green-field; all technology choices must be made by the SOFA participants. The customer himself does not have a software engineering background.

Project gamma: In project gamma, the students were given the task to extend an existing standalone client application for geo-marketing in the areas of sales, marketing and controlling. The extension must be capable of displaying different location based information in a geographical map. Data must be retrieved from a central XML repository, which can be queried using a proprietary object-oriented query language. The customer of the project is a geo-marketing consultancy in Germany, which, among others, maintains its own geo-marketing software tools.

Project delta: Project delta is a green-field project. The customer is a traditional family-operated rose-growing company in the Netherlands, who operates mainly on the international container market. The goal of the SOFA project team is the development of an addition to the customer's enterprise resource planning (ERP) system, which is capable of processing information gathered from RFID tags, which will be attached to the different types of rose transportation devices, repositories and gates. Apart from scanning RFID tags, data needs to be gathered using a web application and synchronized with an existing data repository.

Note that two of the projects require software systems to be developed from scratch (projects beta and delta), while the remaining two projects are evolutionary in nature. These characteristics were considered when assigning the projects to either the decision view group, or the comparison group, i.e. each of the study groups has one green-field and one brown-field project. Please refer to Section 2.4.3 for the details of the study group assignment.

2.4.2. Subjects and sampling

Using students in empirical studies is a sensitive issue that obliges researchers to take a number of ethical and epistemological factors into account. On the one hand, studies with students are often criticized for not being generalizable (Svahnberg et al., 2008); on the other hand, researchers should make sure that the study has as much pedagogical value for the participating students as possible. To make sure that these factors were sufficiently taken into account, Carver et al.'s checklist for conducting empirical studies with students (Carver et al., 2010) was used as a guideline for the design of this study. In the following, we list all items of this checklist together with a brief explanation on how the checklist item was considered:

1. **Ensure adequate integration of the study into the course topics** – The research goal was to study the effect of decision documentation on the design process of junior software designers. The educational goal of the study was twofold: The students should become aware of problems in their decision making processes, and be provided with concrete ways to tackle these problems. The main educational goal of the SOFA project is to familiarize students with realistic software projects, in which they have to make all design decisions themselves (i.e. without assistance by lecturers), communicate with the customer, and take over responsibility for their end-products. Therefore, by conducting the study in a course, in which the students have to work in project teams to solve a real-world case, the study was properly integrated into the course topic.
2. **Integrate the study timeline with the course schedule** – The timeline for the study was explicitly planned according to the start of the SOFA project. The first seven weeks of the project were observed, because naturally, the most design decisions had to be made in the first half of the SOFA semester, while the second part would be primarily spent on programming and report writing.
3. **Reuse artifacts and tools where appropriate** – The tools and artifacts gathered in the study were all part of the SOFA course. Apart from decision views, the students did not have to use additional tools or create additional artifacts for the purpose of the study.
4. **Write up a protocol and have it reviewed** – A study protocol was written before the study and reviewed among the authors in multiple iterations. In addition, the study was discussed with the five lecturers of the course to make sure that it aligns with the course and makes no unrealistic assumptions.
5. **Obtain subjects' permission for their participation in the study** – At the beginning of the SOFA course, the students were informed about the plan to conduct an empirical study in the context of the module. In particular we explained which data we would collect and assured them that no information would be shared with their course lecturers. The students were not informed about the concrete goal of the study, because this could have biased the results. They were given the opportunity to withdraw from the study without giving further reasons, e.g. by sending an e-mail to one of the researchers.
6. **Set subject expectations** – Prior to the study, the students were informed about the purpose of the study, the time they would need to invest and the benefits they can expect from the study, i.e. information about and suggestions for improving their design processes.
7. **Document information about the experimental context in detail** – This research report contains detailed information about the experience of the subjects, the nature of the SOFA course, and the concrete projects run as part of the SOFA.
8. **Implement policies for controlling/monitoring the experimental variables** – The study variables, as well as the data collection methods and data sources used to monitor the variables are described in detail in this study report.
9. **Plan follow-up activities** – At the end of the semester, one of the researchers presented the preliminary results of the study to the students. On this occasion, the students were informed about the concrete goals of the study and the research conjectures, namely that the project teams who documented decision views would be expected to follow a more rational decision making process. The study design was also discussed with the students as well as potential threats to validity. That way, the students also learned something about conducting case studies, which was especially interesting, because the students had been following a course on applied research methods as part of their curriculum.
10. **Build or update a lab package** The collected data was assembled in a study database (as proposed in Yin, 2009), which was used as a basis for the analysis and prepared for reuse in future studies.

The participants of the study were selected using convenience sampling (Given, 2008); all students who took the SOFA course in the winter semester 2011/2012 were invited to participate. None of them refused. In total, 21 students took part in the study. The researchers were not given the opportunity to intervene in the

[1] A company that acts as a broker between restaurant owners and the searched cooking personal.

Table 1
Descriptive statistics used for assigning projects to study groups.

	Decision view group		Comparison group	
	Project alpha	Project beta	Project gamma	Project delta
Avg. programming experience (months)	49.2	56.8	68.4	82
Avg. design exp (months)	32.2	50.33	43	74
Avg. industrial exp. (months)	7	25.18	31.83	19.28
Primary domain(s)	Web applications	Web applications	Marketing	Web applications, logistics
Avg. exp. primary domain(s) (months)	13.4	31.2	13.33	32.5
Median difficulty of project	3	4	3	3
Greenfield (GF) or brownfield (BF)	BF	GF	BF	GF

assignment of students to one of the four SOFA projects introduced before. Each student chose a project based on personal interests.

2.4.3. Assignment of projects to study groups

As mentioned before, we were not given the opportunity to assign students to the four available projects, hence we could only assign the four project teams to the study groups (i.e. decision view group or comparison group) in a way that both study groups were balanced with respect to the relevant case variables, as far as possible. The following case variables were taken into consideration. First of all, the two study groups should be balanced with respect to green-field and brown-field projects (see Section 2.4.1). Second, the difficulty of the project tasks should be comparable in both study groups (CaseVar6, Section 2.3). Finally, the students' industrial experience, as well as previous experience regarding programming, architecture, and the domain of the project should be balanced in both study groups as far as possible (CaseVar2,3,4, and 7, Section 2.3).

Table 1 shows descriptive statistics for the variables used as a basis for the assignment of SOFA project teams to study groups.[2] The data was gathered using a web questionnaire. For the estimation of the difficulty of the four projects, we used the procedure described for CaseVar6 in Section 2.3. See Table B.6 for the detailed ratings given by the four lecturers and the researchers. Table 1 shows only domains that were selected by the majority of students in each project (referred to as primary domain). In the case of project delta, two domains were equally often selected. The average domain experience refers only to the primary domain; in case of project delta, the average was calculated from both primary domains. The most important project characteristic for the study group division was the current state of the software project; i.e. a new project starting from scratch (green-field project), or an existing project that is further developed (brown-field project). Thus, we had to assign one brown-field project to each of the study groups. Both brown-field projects were similar with respect to domain experience and difficulty, but the members of project gamma were more experienced regarding programming, design, and industrial experience; therefore, we decided to assign project gamma to the comparison group. This was mainly to exclude the previous experience as a confounding factor with respect to the rationality of the decision making processes; if we had assigned the more experienced project to the decision view group, the more rational decision making process could have resulted from the previous experience, rather than from the documentation of decision views.

Between the two green-field projects, we decided to assign project beta to the decision view group and project delta to the comparison group. In this case, the advance of programming and software design experience on the side of project delta compensates the difference of approximately six months with respect to industrial experience.

[2] More detailed statistics about the variables can be found in Appendix B

2.4.4. Scrum process followed

The degree, to which the project teams actually followed the Scrum process (CaseVar8) could not be taken into consideration for the study group assignment, because we had to wait for the end of the case study to do this analysis.

However, we found that none of the four teams rigorously adopted the Scrum process. Although all of them organized the development in sprints (2 weeks in project delta; 3 weeks in the other projects), assigned the role of the scrum master to a team member, and created backlogs, other essential parts of Scrum were neglected. Most importantly, none of the teams adequately filled in the role of the product owner.

Appendix C more elaborately describes, how far each of the project teams adhered to roles, meetings, and artifacts defined in the Scrum process. Our findings suggest that the recommendation of Scrum as a development method did not have a significant impact on the decision making process of the project teams. Nevertheless, we discuss the Scrum recommendation as a potential threat to the validity of our results in Section 4.

2.5. Instrumentation and data collection procedures

The response variables were measured mainly in the first seven working weeks. A final round of focus groups with all four project teams was conducted at the end of the SOFA semester in January 2012. In the following, we describe the instrumentation and data collection procedures used.

2.5.1. Instrumentation

The decision view group received a two-hour training on creating the decision views additionally to written guidelines and examples. The training and the reading material covered the following two topics:

Architecture decisions: The students learned how to identify architecture decisions. While they immediately understood that architectural design is basically a decision-making process, some of the students did not know how to differentiate between design decisions in general, and architecture decisions in particular. We told them that, as a rule of thumb, a decision is architectural, if its impact is not locally limited to a single component, for instance, but if it affects larger parts of the system's structure or the overall quality attributes of the system. We also gave them some examples of architecture decisions, e.g. the choice of a middleware platform, a framework, or a database management system; or the decision to apply an architectural pattern. We advised the students to always treat a decision as architectural, in cases where they were in doubt.
Decision viewpoints: All decision viewpoints were introduced to the students. Apart from the formal viewpoint specifications (van Heesch et al., 2012), we discussed examples for all types of views. To enable the students to document views themselves, we let them create decision views from scratch. During this process, an instructor corrected mistakes

Table 2
Mapping of variables to data collection methods.

	Questionnarie	Work diaries	Documentation analysis	Focus groups
Resp1: Identification of ASRs		X	X	X
Resp2: Requirements negotiation		X	X	X
Resp3: Prioritization of Requirements			X	X
Resp4: Documentation of Requirements			X	X
Resp5: Discovery of design options		X	X	X
Resp6: Balancing advant. and disadvant.		X	X	
Resp7: Discuss mult. options in comb		X	X	X
Resp8: Avoid complexity		X	X	X
Resp9: Validate options against ASRs		X	X	X
Resp10: Prototyping options		X	X	X
Resp11: Evaluation of arch. as a whole		X	X	X
CaseVars2–4: Previous experience	X			
CaseVar5: Project domains	X		X	
CaseVar6: Difficulty of the project		X	X	
CaseVar7: Experience in domain	X			
CaseVar8: Scrum process followed		X	X	X

and answered questions. The presentation slides used to introduce decision viewpoints to the students are available online: http://www.vanheesch.net/training_decision_viewpoints.pdf.

Additionally to the training and guidelines, the members of the decision group were provided with an MS Word template for the detail view, an MS Excel template for the decision forces view, and a Visual Paradigm template containing an example of a relationship view. All members of the decision view project teams were obliged not to talk to the members of the other two project teams about decision views. The comparison group was not informed about decision views at all.

2.5.2. Data collection procedures

In the following, we describe the data collection methods used, the data sources, and their mapping to the study variables. In qualitative research, it is important to develop ideas not only based on one data source using one specific data collection method. Triangulation of data sources and data collection methods has been a good practice for qualitative researchers to make sure that there are multiple forms of evidence to back up a conclusion rather than single data points or very few incidents (Creswell and Miller, 2000; Lethbridge et al., 2005; Yin, 2009). Patton differentiates four types of triangulations (Patton, 2002): (a) data source triangulation, (b) investigator triangulation, (c) theory triangulation, and (d) methodological triangulation. In this study, we apply data source triangulation (collecting data from more than one source) and methodological triangulation (collecting data using different methods). Our motivation is to have at least two data sources, and their corresponding data collection methods, for all conclusions drawn from the collected evidence.

We use the data collection classification scheme from Lethbridge et al. (2005) to describe the methods used in this case study:

- **Questionnaire**: At the beginning of the study, the students filled in a questionnaire to gather information about the nature of the SOFA project they chose, their previous experience in software engineering activities, and their experience in the domain of the chosen SOFA project.
- **Work diaries**: As part of the SOFA course, the students had to create daily work diaries, in which they document their process, the decisions they made, and the tasks they identified. These diaries (also referred to as team minutes in the remainder of this paper) were made available to the researchers.
- **Documentation analysis** (study of work artifacts): The Subversion and Trac servers, which all project teams had to use, were accessible for the researchers. All working artifacts that were checked in by the different project teams (e.g. requirements and design documents, minutes, source code, and the decision views) were collected and analyzed. In addition, one person of every project team was responsible for taking pictures of all whiteboard sketches the project teams created.
- **Focus groups**: Weekly focus groups (30–60 min) with each of the project teams about the design process and the decisions made (audio recorded and transcribed). In the focus groups, the participants were encouraged to talk freely about their design, the progress they made, and the process they have been following. Additionally, in the decision view project teams, the documented decision views were also discussed. In the comparison group projects, the participants were interviewed without explicitly mentioning the notion of architecture decisions. In addition to the weekly focus groups, a final round of focus groups was conducted with each of the project teams at the end of the SOFA semester. Furthermore, the focus group moderator took field notes, in which he noted down impressions gained during the focus groups. Field notes are important to complement the audio recordings, as some important information is non-auditory, e.g. supporting a teammate's comment by nodding, or strong opposition expressed only in body language. Field notes can also be used to write down initial ideas about the project process that can serve as focus points during the data analysis, e.g. "It seems that project alpha did not align the user stories with the functional requirements gathered before".

The field notes and all collected data were stored in a digital study database.

Table 2 shows a mapping of study variables to data collection methods. Note that there are at least two data sources for each response variable.

2.6. Analysis procedure

The gathered data was analyzed qualitatively using grounded theory (Glaser and Strauss, 1967). In grounded theory, theories are developed by systematically analyzing the collected data, and constantly comparing findings to the previous conjectures. It is thus a very labor-intensive, iterative process, in which theories evolve slowly and are always grounded in the collected data. In recent years, the use of grounded theory in software engineering has become acceptable (Urquhart et al., 2010; Adolph et al., 2011); conceivably, because contemporary research in the field seeks to involve more exploratory research, rather than relying on purely confirmatory studies.

Table 3
Categories.

Code	Category	Eval.
Cat1	Systematically searched for multiple design options.	+
Cat2	Conducted research to identify design options.	+
Cat3	Most design options are technology related.	
Cat4	Followed a reuse over reimplementation strategy.	
Cat5	Research tasks regarding design options for a decision point were divided among the group.	
Cat6	Developed overall vision of the architecture to identify decision points.	+
Cat7	Always chose first viable solution.	–
Cat8	Most gathered requirements are functional.	–
Cat9	Non-functional requirements were not actively elicited.	–
Cat10	Actively explored the functional problem space.	+
Cat11	No explicit distinction between architecturally relevant requirements and other requirements.	–
Cat12	Actively involved to understand the business domain.	+
Cat13	The group tried to install and run the existing software as a first step in the analysis.	
Cat14	Multiple types of documentation used for requirements.	–
Cat15	Systematically clarified vague requirements with customer.	+
Cat16	Responsibility for describing requirements is silently transferred to the customer.	–
Cat17	Requirements slowly emerged during the design phase.	
Cat18	No clear separation between requirements and resulting design or implementation tasks.	–
Cat19	Quality attribute requirements were not documented.	–
Cat20	Team does not gain a collective understanding of the requirements.	–
Cat21	Scrutinized requirements with respect to feasibility and usefulness.	+
Cat22	Requirements were not called into question.	–
Cat23	Proposed additional requirements.	+
Cat24	Negotiated blocking requirements.	+
Cat25	Challenging requirements were prioritized.	+
Cat26	Requirements are addressed in no recognizable order.	
Cat27	Requirements were not described well. Single words or brief statements used without explanation.	–
Cat28	Explicitly discussed pros and cons of all major design decisions.	+
Cat29	Conducted research to find arguments in favor of and against design options.	+
Cat30	Group members challenge each others arguments a lot.	+
Cat31	Decisions are mostly made without explicit reasoning.	–
Cat32	Most decisions are not discussed in the group.	–
Cat33	Technological dependencies were systematically explored before making decisions.	+
Cat34	Many technological decisions were made in combination.	+
Cat35	Avoiding complexity was an explicit goal of the group.	+
Cat36	Validated technology options against ASRS.	+
Cat37	No indicators for an explicit consideration of ASRs when making decisions.	–
Cat38	Were aware that trade-offs could be necessary.	+
Cat39	Made trade-offs between multiple requirements (very rarely).	+
Cat40	Used prototypes to understand technological options.	+
Cat41	Prototypes mainly used to learn how the technology can be used.	
Cat42	Prototypes were used to estimate the influence of a design option on quality attribute requirements.	+
Cat43	Permanently maintained an overview over the complete system.	+
Cat44	Architecture was not evaluated as a whole.	–

One might suggest that grounded theory is not appropriate in cases where researchers seek to verify a research hypothesis (Urquhart et al., 2010), i.e. the research is confirmative rather than exploratory. In this particular case, however, despite of the formulated research conjecture, the research is fundamentally exploratory: we attempt to gain a broad understanding of how the students design software and how the documentation of different decision views during the design process influences the design activities. This explorative part of the study is a prerequisite for the comparison of the two study groups.

2.6.1. Qualitative analysis approach used

In the following, we describe the qualitative analysis procedure as applied in this study. The detailed description allows other researchers to assess the quality of the analysis process, or to adopt it in own studies. Fig. 1 shows a UML class diagram summarizing the conceptual entities of the qualitative analysis and their relationships respectively; they will be discussed in the following.

Step 1 - Filter study documents: In the first step, we browsed all documents that had been collected in the study database during the course of the study. Besides transcripts, field notes, and minutes, this process included all documents that were created by the SOFA project teams and uploaded to their respective subversion repository or Trac wiki.

If the content did not have any relation to the decision-making process, the document was excluded; all other documents were taken to the next steps, described below. In total, 401 documents were browsed in step one, 254 of which were found to be relevant. The excluded documents contained documentation of used third party software, or bash scripts and latex templates, for instance.

Step 2 - Normalization: In the next step, the chosen documents from step one were normalized: each file was converted to PDF, and renamed to express the name of the SOFA project, the type of file, the original file extension, and the date at which it was downloaded. The result of this step was a number of PDFs assigned clearly to one of the four SOFA projects.

Step 3 - Coding: In step three, the documents from each SOFA project were coded. During this procedure, the documents were carefully studied and each phrase, sentence, or paragraph that indicated a certain behavior (called *indicator* in grounded theory literature Strauss, 1987) was labeled with a code. This approach to coding is originally referred to as *open coding* (Corbin and Strauss, 2008), used to generate the concepts that become the basis for further analysis (see step 4).

As opposed to other researchers, who suggest to assign one-word codes to express indicators (e.g. Adolph et al., 2011), we

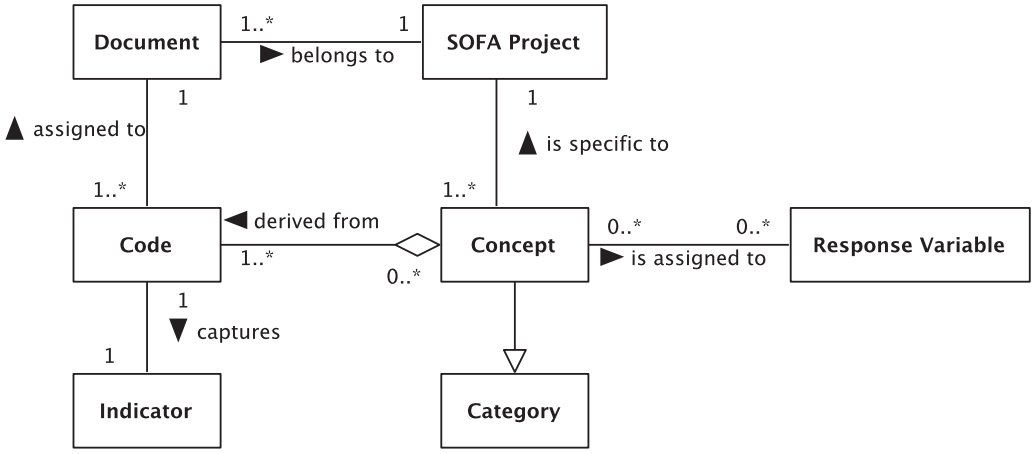


Fig. 1. Conceptual model of grounded theory entities.

chose to use brief statements as codes. We made this decision, because we experienced that finding single words to clearly express an indicator is extremely challenging and forces the analyst to read large passages over and over again, when comparing codes to previously assigned codes. Brief statements allow to be much more expressive. We used PDF annotations to assign the codes. That way the codes are shown next to the text without disturbing the flow of reading. PDF annotations have the additional advantage that codes can easily be revised and different analysts can assign different codes.

To collect the codes from the documents and to support the constant comparison process, we developed a software that registers documents, collects all codes and stores them in a study database using the model shown in Fig. 1 as domain model. The constant comparative method in grounded theory obliges the researcher to frequently step back to analyze all collected codes and to compare new codes to existing ones to develop a theory. The tool we developed supports this process, as previously assigned codes from other documents are permanently shown to the analyst while coding additional documents. This enables the analyst to identify commonalities in codes, which is useful for discovering concepts.

Step 4 - Identify concepts: Steps three and four were repeated in multiple iterations when analyzing the documents of one SOFA project. In step four, the previously gathered codes were compared to identify common concepts. Here, a concept is a representation of a pattern of behavior, suggested by a set of indicators, which on their part are captured using codes (Adolph et al., 2011). The concepts were assigned using the previously mentioned software, we developed. During the analysis procedure, the concepts slowly evolved; they had to be revised regularly after additional documents had been analyzed.

Step 5 - Assign concepts to variables: After finishing the coding and the declaration of concepts, each concept was assigned to one or more response variables.

Step 6 - Classify concepts into general categories: After finishing steps one to five, we had defined a set of concepts describing the behavior of each project team with respect to the response variables. The concepts are specific to projects, i.e. they summarize multiple codes from the documents of one project team. In order to compare the results from the different project teams, we analyzed the concepts and classified them into categories. A category, in our understanding, is a project independent abstraction of one or more concepts from potentially different projects. This is in line with Glaser, who describes a category as a concept used on a higher level of abstraction (Glaser, 1998). Fig. 2 illustrates the relationship between categories and project-specific concepts. The categories, defined in this study, can be found in Table 3.

3. Analysis and interpretation

In this section, we present the results of the qualitative analysis with respect to commonalities and differences between the projects in the two study groups. The section is organized according to the response variables. The codes and concepts, which are the result of steps three and four in the analysis, are not listed here for reasons of space; in total, more than 620 codes were assigned to the various documents resulting in 165 concepts. Table 3 lists the categories identified in Step 6 of the analysis. The last column ("Eval.")

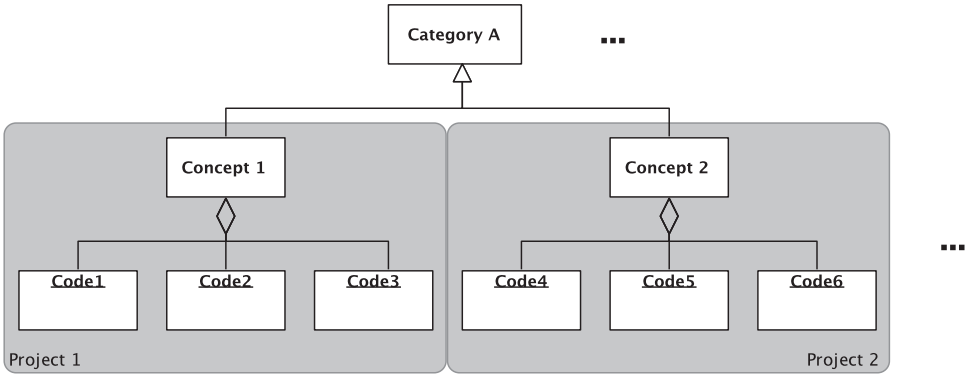


Fig. 2. Comparing projects using categories of concepts.

Table 4
Response variables and categories observed in projects.

Response variable	Category	Eval.	Decision view group		Comparison group	
			Alpha	Beta	Gamma	Delta
Resp1: Identification of ASRs	Cat8	–		X	X	X
	Cat9	–	X	X	X	X
	Cat11	–	X	X	X	X
	Cat16	–			X	X
	Cat18	–	X		X	
	Cat19	–	X		X	X
	Cat20	–				X
	Cat10	+		X		
	Cat12	+		X		X
	Cat15	+		X		
	Cat13		X			
	Cat17		X			X
Resp2: Requirements negotiation	Cat22	–			X	X
	Cat21	+		X		
	Cat23	+		X		
	Cat24	+	X	X		
Resp3: Prioritization of requirements	Cat25	+	X	X		
	Cat26				X	X
Resp4: Documentation of requirements	Cat14	–	X	X	X	X
	Cat27	–	X	X	X	X
Resp5: Discovery of design options	Cat7	–			X	X
	Cat1	+	X	X		
	Cat2	+	X	X		
	Cat6	+	X	X		
	Cat3		X	X		
	Cat4			X		
	Cat5		X	X		
Resp6: Balancing pros and cons of design options	Cat31	–			X	X
	Cat32	–			X	X
	Cat28	+	X	X		
	Cat29	+	X	X		
	Cat30	+	X	X		
Resp7: Discussion of multiple design options in combination	Cat33	+	X	X		
	Cat34	+	X	X		
Resp8: Avoidance of complexity	Cat35	+	X	X		
Resp9: Validation of design options against the ASRs	Cat37	–			X	X
	Cat36	+	X	X		
	Cat38	+	X	X		
	Cat39	+	X			
Resp10: Prototyping design options	Cat40	+	X	X	X	X
	Cat42	+		X		
	Cat41		X		X	X
Resp11: Evaluation of architecture as a whole	Cat44	–	X	X	X	X
	Cat43	+	X	X		

indicates, whether we regard the observed behavior as positive (+), neutral (empty column), or negative (−). In the remainder of this section, we qualitatively describe the students' decision making process using the response variables and categories.

Table 4 shows the categories as assigned to the four projects and response variables respectively. It was taken as a basis for the subsequent analysis. Cases, in which a category was assigned to each of the four projects, are regarded as a commonality; cases, in which a category was assigned to the two projects within one study group, but not in the two projects from the other study group, are regarded as a difference. In principle, these differences do not necessarily result from the fact that the two projects in the decision view study group used decision views. Therefore, for all noticed differences, we discuss how far the difference results from the decision view creation, i.e. how far the decision view creation is the cause for the differences. Cases, in which a category was assigned to a single project only, or cases in which a category was assigned to one project in the decision view group and one project in the comparison group are not discussed, because they do not allow drawing conclusions with respect to the impact of using the decision viewpoints on the design activities. Column "Eval." repeats the evaluation of the respective category (i.e. positive, neutral, or negative) from Table 3. For each response variable, we list the assigned

Table 5
Summary of findings.

Variable	Variable description	Dec. view supp.
Resp1	Identification of ASRs	+
Resp2	Requirements negotiation	+
Resp3	Prioritization of requirements	+
Resp4	Documentation of requirements	∼
Resp5	Discovery of design options	++
Resp6	Balancing advant. and disadvant.	++
Resp7	Discuss mult.options in comb.	++
Resp8	Avoid complexity	∼
Resp9	Validate options against ASRs	+
Resp10	Prototyping options	∼
Resp11	Evaluation of arch. as a whole	+

++, strong support; +, partial support; ∼, no support.

categories in the following order: negative categories, positive categories, and neutral categories.

3.1. Resp1 – Identification of ASRs

None of the four project teams actively elicited non-functional requirements (Cat9). In some cases, the students even ignored hints given by the customer with respect to quality attribute requirements. All project teams focused on functional requirements, which were mainly understood as use cases, or user stories.

Generally, the four project teams also did not make an explicit distinction between requirements in general, and architecturally relevant requirements (Cat11). The only exception were the forces views, which triggered the two project teams in the decision view group to select only those requirements that were architecturally relevant. In all other documents containing requirements, this distinction was not evident.

While the two project teams in the decision view group actively approached the customer multiple times to elicit and clarify requirements, the two project teams in the comparison group transferred the responsibility for defining requirements completely to the customer (Cat16). They took no effort to elicit requirements additionally to an initial list of requirements they received from the customer. In case of project delta, the students neglected requirements elicitation, although the customer explicitly told them in the beginning that he expected them to do a thorough analysis of the project domain and the resulting requirements. Both project teams in the comparison group also took no effort to clarify requirements they had not understood well; they rather speculated about their meaning internally during the project meetings. The difference regarding the active elicitation of requirements was potentially caused by the decision forces view, which requires students to actively reflect on requirements and other forces that influence their design decisions.

3.2. Resp2 – Requirements negotiation

There is a notable difference on how the two study groups negotiate requirements. The two project teams in the comparison group did not question any requirements (Cat22), the project teams in the decision view group actively went into requirement discussions with the customers. Both project teams in the decision view negotiated requirements they experienced as unnecessarily constraining or even blocking (Cat24).

To give an example, the customer obliged project team alpha to use JDK 1.5, a rather old Java Development Kit, because one of the third party libraries used by the customer was not compatible with newer Java versions. This technological constraint turned out to have a huge impact on the design options that could be taken into consideration; particularly on the choice of the enterprise service bus (ESB) technology. Most recent ESB implementations require a JDK greater than 1.5, which would have forced the students to rely on older implementations and at the same time older versions of the Java enterprise edition (JEE). This, however, would have prevented the usage of frameworks, which require newer versions of JEE. To understand the true dimensions of the JDK 1.5 constraint, the students created a decision relationship view showing the technological choices they would have made without the constraint, and the technological choices they could make regarding the constraint (see Fig. 3). Using this relationship view, the customer could be convinced to drop the constraint.

Apart from this example, the decision forces view seemed to create a much more critical attitude in the decision view group towards the requirements compared to the comparison group, because it forced the two project teams in the decision view group to actively reflect on requirements and the design options that can possibly satisfy them. The two project teams in the comparison group took the requirements for granted (i.e. they did not question them). To make matters worse, they also did not make sure that the decisions they made were consistent with the requirements, as we will explain later.

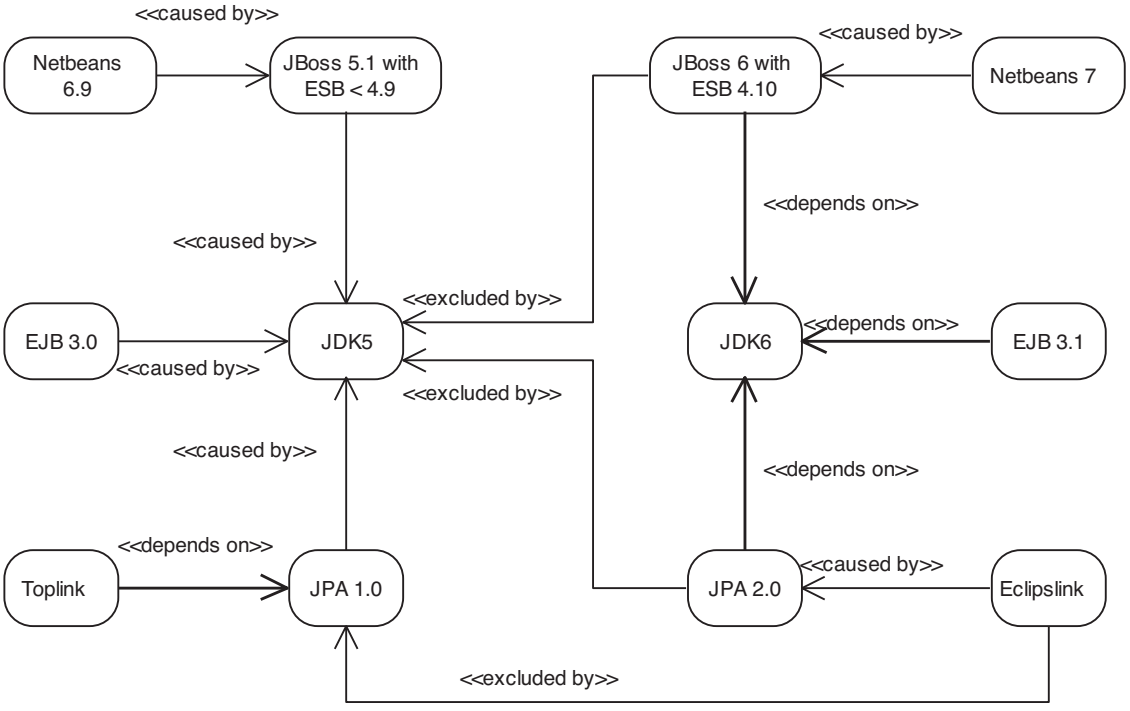


Fig. 3. Relationship view created by project team alpha to understand the impact of a technological constraint.

3.3. Resp3 – Prioritization of requirements

Another difference between the decision view group and the comparison group was observed with respect to the prioritization of requirements. The project teams in the decision view group prioritized requirements (Cat25). Requirements, recognized as being challenging or very important, were given a higher priority than other requirements. As an example, project team alpha immediately started searching for technological design options to realize a service oriented architecture, while other requirements regarding the realization of a full-text search and a tag cloud were given such a low priority that they could not be implemented until the end of the project. Project team beta put priority on finding a web framework that would allow to add and change content easily. Requirements regarding social media and third party payment provider integration, in accordance with the customer, were given a lower priority.

As opposed to the decision view group, the project teams in the comparison group did not address requirements in a recognizable order (Cat26). Although in both projects, some of the requirements were clearly more challenging than others, these requirements were not given a higher priority. As an example, in project delta, the students had to make sure that RFID scanners would work partially in an unfriendly environment (outside, exposed to dirt, whether, heat, and cold temperatures), using different types of available networks (e.g. LAN, WIFI, and GPRS) without losing data. Until the end of the project, the students ignored these requirements. In project gamma, requirements were chosen based on the personal interests of project members, instead of estimating their importance systematically.

3.4. Resp4 – Documentation of requirements

In all four projects, we observed that different types of documentation were used to capture requirements (Cat14). In addition, none of the project teams assembled requirements in a central place. The project teams alpha and delta captured initial requirement statements, made by the customer, in a word document; all four teams documented some functional requirements using use cases stored at different places in the projects' repositories; project teams beta and gamma additionally documented user stories (a Scrum-specific way of documenting customer requirements) in the Trac systems. In the decision view projects, additional requirements and forces were captured in the forces views. When comparing the requirement statements made in the different documents, inconsistencies were found in all four projects. For instance, some use cases documented by projects beta and gamma were not documented as user stories, whereas other functional requirements only existed in the form of user stories, but not in the form of use cases.

Another phenomenon observed in all four projects is the fact that the different types of requirements documents were not revised or updated any more. This is a strong indicator for the lack of a thorough requirements management process. In the focus groups, all four project teams acknowledged that requirements kept changing, but they also admitted that existing requirements documentation was not systematically adapted to those changes. Therefore, the existing requirements documentation was quickly outdated. As a consequence, the students in all four projects were not able to gain a holistic understanding of all relevant requirements; nor could they systematically regard all requirements in the design process.

Another major issue commonly identified in all projects was the fact that the documented requirements did not have the SMART (Mannion and Keepence, 1995) characteristics. The students mainly used brief statements, sometimes only single words, to express requirements (Cat27). An example from project alpha, which was not further elaborated, is "Fulltext search should be possible". An example from project delta is "assigning products to Ferro/Pallet Tags", or simply "Store data". Our initial intention, to analyze the documented requirements with respect to how SMART they are was dropped, because the quality of the requirements documentation was so low that a further analysis for the purpose of comparison would not have made sense.

The findings regarding Resp4 suggest that the decision viewpoints did not help the students to systematically document and manage requirements.

3.5. Resp5 – Discovery of design options

As opposed to the comparison group, the decision view project teams systematically searched for multiple design options (Cat1). To identify design options, the two project teams conducted research (Cat2) using the Internet. Already at an early stage, the decision view project teams developed a vision of the overall architecture to identify decision points (Cat6, see Figs. D.9 and D.10 for two examples) and then assigned different team members to conduct research regarding design options for a specific decision point (Cat5). Both project teams considered at least two design options for each major decision point. It was evident that most design options were technology-related (Cat3). Only in rare cases, the project teams considered the use of a design or architectural pattern, for instance.

In contrast to the decision view group, the two project teams in the comparison group did not systematically search for design options before making decisions. Even when being asked directly about considered alternatives, they acknowledged that they only searched until they found a viable solution (Cat7) and then moved on to another decision. In case of project delta, the students used mainly trial and error. Only when a chosen design option turned out to be the wrong choice during the implementation, they hastily searched for an alternative.

3.6. Resp6 – Balancing advantages and disadvantages of design options

A significant difference between the two study groups is how design options were analyzed and compared. The project teams in the decision view group heavily used the forces view to explicitly discuss pros and cons of major design decisions (Cat28). The project teams created forces views to support single decisions, and kept revising one central forces view with all major design decisions. Fig. 4 shows a forces view created by project team beta to support the choices of a programming language and a database management system.

Apart from requirements statements, the decision view group also explicitly discussed other decision forces like learnability of technologies or previous experience of the project members. In all cases, the forces view was used as a means to discuss and capture design choices and their arguments.

The decision view project teams systematically filled knowledge gaps that became apparent during the discussions, by conducting research on particular design options (Cat29). Each team member presented the results of the design options research to the other members before a decision was made. This often led to intense discussions, in which the team members heavily challenged each others arguments (Cat30). The students appreciated that decision forces helped to spread knowledge about individual design options better among the different project members and that it provides a framework for discussing options systematically. In the focus groups, both project teams in the decision view group acknowledged that the decision views had a huge impact on the discussions of design options. A member of project team alpha said: "If you don't have the view, then you might also see alternatives, but if you have

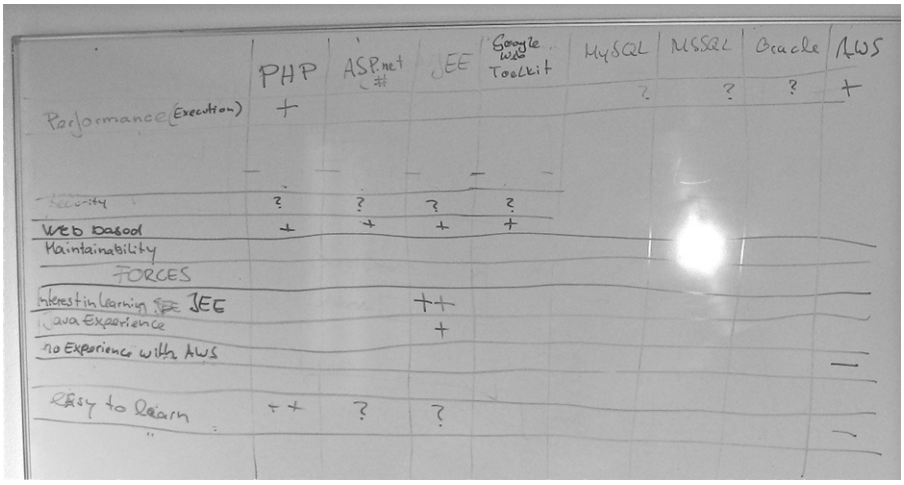


Fig. 4. Partial forces view created by project team beta.

experience in a solution then you will choose this one. But with the (forces) view, you are forced to think about which one is really better.". In accordance, a member of project team beta stated: "I think the fact that we had to document decisions and our decision making had an influence on the seriousness that we handled decisions. So if there wouldn't have been these views, then maybe we would have been faster in decision making; just say ok that works, so let's take it".

As opposed to the decision view project teams, the two comparison project teams made decisions mostly without explicit reasoning (Cat31). The project teams searched for viable solutions and applied them without systematically discussing their advantages and disadvantages. When being asked about the rationale behind specific decisions, the students gave the impression that they started thinking about pros and cons just in the moment when the question was asked; however this is the impression of the focus group moderator and not a fact. In case of project delta, in many cases, the members were not able to provide any rationale for major design decisions. Finally, judging from the focus groups and the team minute meetings, the comparison project teams did not discuss design decisions in the group (Cat32). Instead, decisions were made by single members and silently accepted by the other team members.

3.7. Resp7 – Discussion of multiple design options in combination

Both project teams in the decision view group used the relationship view to systematically explore technological dependencies before making decisions (Cat33). As mentioned above, the project teams created partial relationship views on the whiteboard to support design discussions. The relationship view showed the students that some decision have a great impact on other decisions. Both project teams spent a lot of effort on understanding these impacts. Many technological decisions were made in combination (Cat34); both project teams not only discussed alternatives for single decisions, but also compared multiple graphs of decisions (multiple combinations of decisions, which as a whole, are alternatives to each other).

The project teams in the comparison group did not discuss multiple design options in combination, but rather made decisions without evaluating impacts on other decisions.

3.8. Resp8 – Avoidance of unnecessary complexity.

Only very few indicators were found that any of the student projects explicitly tried to avoid unnecessary complexity in their design. Both of the project teams in the decision view group, however, stated that avoiding unnecessary complexity was an explicit goal within the project (Cat35). When being asked how unnecessary complexity could be avoided, project team alpha stated that they tried to minimize the usage of third party libraries, particularly libraries that come with a lot of unneeded functionality. Project team beta explained that they reduced unnecessary complexity by trying to find a middleware framework that provides a great part of the needed functionality out of the box. The statements of both project teams could be verified by analyzing the forces view and the architectural design; however, no other examples for explicit avoidance of unnecessary complexity could be found.

The project teams in the comparison group did not explicitly avoid unnecessary complexity. However, there is not enough evidence to show that the use of the decision viewpoints help them to avoid unnecessary complexity.

3.9. Resp9 – Validation of design options against the ASRs

As mentioned for Resp4 already, the documentation of requirements was weak in all four projects. However, the decision view project teams at least considered architecture significant requirements when making technological choices using the forces view (Cat36). In the comparison group, no evidence was found that architecture significant requirements were considered when making decisions (Cat37): design decisions were made without systematically identifying alternatives, while there were no indicators that design options were validated against ASRs. The project teams in the decision view group were aware of the fact that in some cases trade-offs between multiple requirements could be necessary (Cat38). Project team alpha used the decision forces view to resolve such situations, but they declared that this happened very rarely (Cat39). Indeed, their forces view showed that the students did not come across many conflicts that had to be resolved.

3.10. Resp10 – Prototyping design options

All four project teams heavily used small prototypes to understand technological options (Cat40). In particular, they created prototypes to understand how technologies (e.g. frameworks or libraries) must be used (Cat41). However, only project team beta systematically created prototypes for the purpose of understanding advantages and disadvantages of multiple alternative design options (Cat42). The other projects, in contrast, created prototypes only after a decision was made. Thus, there is no observable influence of the usage of decision views on prototyping.

3.11. Resp11 – Evaluation of the architecture as a whole

Apart from the discussion of multiple design options addressing identical problems (e.g. database management systems to be used as a central datastore), none of the four project teams explicitly evaluated the architecture as a whole (Cat44). Nevertheless, when being confronted with this issue, the two project teams in the decision view group mentioned that the decision views allowed them to permanently maintain an overview over the current state of the architecture (Cat43). In particular, they mentioned that the forces view always gave them a good estimate over the coverage of the requirements, that's why they (falsely) assumed that a dedicated architecture evaluation session was not necessary.

3.12. Variations of decision view usage

Apart from the findings reported above, we learned that the students in the decision view projects had divergent preferences regarding specific viewpoints. As described in Section 2.4.3, project alpha was a brown-field project, whereas project beta was a green-field. Although the students in project alpha also appreciated the decision forces viewpoint, they saw the most value in the decision relationship viewpoint, because it helped them to analyze and understand technological dependencies. Taking over an existing software project requires software designers to understand the architecture as-is, before they can make any significant changes. Apparently, the relationship viewpoint helped the students in the brown-field project to analyze and document the system as-is; moreover it helped them to resolve a blocking technical constraint, which had a huge impact on multiple technological design options.

Project team beta, the green-field project, experienced the forces viewpoint as the greatest help in the project, although they also made vast usage of the relationship viewpoint. In green-field projects, the solution space is widely open in the beginning. The decisions made in this project stage are highly important and fundamental to the entire system. The decision forces viewpoint turned out to be a useful support for the students to make decisions based on solid argumentation using an agreed-upon evaluation scheme. It gave them more confidence that the decisions they made were the right choices among the available design options.

Even though the relationship and forces viewpoints were very well received by the students, both project teams expressed their discontent about documenting the decision detail view. They experienced documenting single decisions using our template as a tedious job that did not have an immediate benefit for the design process. The same finding had been made by other researchers in the past (e.g. Harrison et al., 2007). Yet, the students acknowledged that the detail views will have a clear benefit for subsequent developers taking over their project.

3.13. Summary of findings

We have found that the decision views provide strong support in the area of solution evaluation and selection, partial support for ASR management, and no support for handling complexity or evaluating the viability of a design option. Table 5 summarizes the findings regarding the decision view support (column *Dec. view supp.*) for particular design activities, based on the analysis of the response variables. Decision views provide strong support for design activities related to *architectural synthesis* and *architectural evaluation*.[3] By far the strongest support was recognized for Resp5, related to the discovery of design options (architectural synthesis). The decision views triggered the two project teams in the decision view group to identify multiple options for decision topics and to thoroughly conduct research to understand these options. The project teams in the comparison group, in contrast, clearly did not attempt to identify multiple options before making decisions; they rather chose the first presumably viable solution they could find.

Concerning architectural evaluation, the impact of decision view was significant for Resp6 (balancing advantages and disadvantages of design options). The decision view groups invested much more efforts in researching, understanding, and discussing advantages and disadvantages of the (candidate) architectural solutions than the comparison group, who made decisions mainly implicitly, without discussing them. In addition, the fact that the decision view project teams consciously made multiple decisions in combination (Resp7, architectural evaluation), shows that the decision views stimulated the students to regard dependencies between decisions and contributed to the understanding of the architecture as a whole.

As Table 5 shows, the use of decision views did not appear to support the students in (1) requirements documentation, (2) avoidance of unnecessary complexity, and (3) prototyping design options. Point (1) was first a surprising result. The decision forces viewpoint and the decision detail viewpoint explicitly require the statement of requirements, which should have caused the students to define requirements more carefully. A discussion of this finding with the students' lecturers at the university showed that the students were not educated in distinguishing between architecturally relevant requirements and other requirements. They were also particularly inexperienced in documenting quality requirements, and business and environmental demands, which have a higher relevance for architecture decisions. Thus, the fact that none of the project teams documented requirements thoroughly suggests that software designers need to be trained in identifying and documenting architecture significant requirements. Additionally, presenting a checklist of the typical forces in specific domains can remind inexperienced designer in carrying out relevant design activities such as documenting ASRs.

Points (2) and (3) are expected. The viewpoints did not help the decision view group to avoid unnecessary complexity (point (2)). Avoiding complexity obliges designers to simplify and optimize a design solution as far as possible. This requires the knowledge of how a solution can be formulated without compromising the requirements. While the decision views help junior designers to evaluate and select good solutions, they cannot create solution options that are beyond the knowledge of the designers. Prototypes are a means to evaluate the influence of a design solution on certain qualities of the target system. The decision forces viewpoint can be used to document the results of these evaluations (i.e. the impact of a force on a certain design option) to support a systematic choice

Table B.6
Estimation of project difficulties (CaseVar6, Likert-scale 1: very simple; 5: very difficult).

Rater	Alpha	Beta	Gamma	Delta
Lecturer 1	2			2
Lecturer 2	4			3
Lecturer 3		4	3	
Lecturer 4		4	3	
Researchers	3	3	2	3
Median	3	4	3	3

[3] Architectural synthesis refers to activities followed to identify candidate architecture solutions for a set of architecture significant requirements; architectural evaluation concerns the validation of those candidate solutions against all architecture significant requirements (Hofmeister et al., 2007).

out of multiple decision alternatives, but it does not prescribe how the evaluations must be done.

Other activities belonging to architectural analysis (Hofmeister et al., 2007) (Resp1–Resp3) were partially supported by the decision views, mainly because the explicit documentation of decisions and forces on decisions raises a general awareness for aspects that need to be taken into consideration when making decisions.

4. Validity

We use the classification scheme proposed by Yin (2009) and Wohlin et al. (2000) to report on potential threats to validity and measures we took against them.

4.1. Construct validity

Construct validity is concerned with the measures used to represent the effect of the method on the study result according to the research conjecture. To ensure validity of the constructs, we identified response variables prior to the study, explained the rationale behind those variables, and assigned them to data collection methods we would use to measure them. Additionally, we used the constant-comparison method to uncover concepts in the qualitative data. The concepts were uncovered from scratch according to their relevance to the study goal, i.e. we had not thought of concepts in advance.

4.2. Internal validity

Internal validity is mainly concerned with the examination of causal relationships between the method used (decision view creation, or ad hoc) and the response variables. Researchers have to make sure that there are no hidden variables that silently affect the investigated objects. The measures we took to mitigate this risk are twofold. First, we carefully defined case variables that could have an influence on the outcome of the study. Second, we used two pairs of projects: one pair that used decision views, and one that did not. The latter reduces the risk of hidden variables. In particular, the following case variables were identified to uncover and deal with potential hidden causalities related to the study results:

- Case variables 2, 3, 4 and 7 concern the previous relevant working experience of the students who took part in the study. To make sure that the experience does not adulterate our results, we made sure that the average experience of the students in the decision view group is at most comparable to the average experience of the comparison group. In most cases the experience of the students in the decision view group was less than that of the comparison group.
- Case variables 5 and 6 concern characteristics of the software projects. For the validity of the results, it is vital that the projects in the decision view groups are comparable to the projects in the comparison group with respect to the factors under study (i.e. the design process). None of the projects is in a domain that would require an adaptation of the design process (e.g. because of special security or safety needs). Instead, all four projects belong to the software engineering and enterprise computing domains. Project delta was additionally assigned to logistics and web application; project gamma to marketing; projects alpha and beta were both additionally assigned to web application. The difficulty of the projects was balanced among the two groups, as described in Section 2.4.3.

For the evaluation of some response variables (Resp5, Resp6, Resp8–10), we analyzed the decision views created by the decision view groups. This bares a potential risk, as sometimes evidence might only be present in decision views, without being visible in other data collected. In these cases, the cause construct could be under-represented in the comparison group, leading the researchers to wrong conclusions. We mitigated this risk by consulting the decision views only in cases, in which the subjects explicitly mention them in other data collected (e.g. minutes or focus group transcripts). This ensures that no conclusions are drawn based on evidence solely visible in decision views.

Another potential threat to validity could stem from the fact that the decision views were discussed in the weekly focus groups. Theoretically, the focus group moderator could have biased the students towards changing the design in a specific way. This, however was avoided to the greatest possible degree. The focus group moderator asked the students to describe their created decision views themselves rather than asking potentially manipulating questions. As a general rule during all contact with students, the researchers did not answer questions or give any advice regarding specific design considerations. However, the students' own reflection on the decision views during the focus groups could have triggered the students to reconsider specific design decisions. This threat cannot be totally invalidated.

A potential threat to validity derives from the fact that the university, in which the study was conducted, recommended the student groups to use Scrum as a development method. The use of Scrum may have influenced the students' architecture decision making process. For instance, Scrum recommends the format of user stories to document functional requirements. Through its association to agile development methods, Scrum may have also influenced the students to generally neglect documentation. This threat is mitigated, because all four projects used Scrum. Thus, the comparison of the decision view group with the comparison group took place at an equal level. Additionally, we did not rely on the students adhering to the Scrum process; instead we carefully observed the entire development process over the study period. Our results, regarding the differences in the decision making process is independent from the used development method. Furthermore, since a Scrum development team is responsible for making (architecture) design decisions themselves in multiple iterations (called sprint in Scrum), the projects are a pertinent target for investigating the decision views' support for iterative decision making. The degree, to which the students actually applied Scrum techniques, is described in Appendix C.

4.3. External validity

External validity concerns the extent, to which the findings of the study are of relevance for other cases. In this particular case, the study was conducted with students, which is always a threat to external validity, because students are lacking professional experience and real life project constrains like short time-to-market and limited budgets. We partially mitigated the latter issue by using external customers and real software projects. The customers of the projects were independent, i.e. they have no relationship to the school or the researchers. Furthermore, the students were in the last year of a four-year Bachelor of Software Engineering degree, i.e. very close to their professional careers, which is why we assume that the results are at least generalizable to the population of inexperienced software designers with a few years of industrial experience.

Another potential threat to the validity of the results derives from the fact that the students in all projects came from the same university of applied sciences. Theoretically, students from other universities, with a different background, could have performed differently. An identical educational background of the subjects in the two study groups, however, is a prerequisite for the comparison

of their design activities. Thus, to completely rule out this potential threat to external validity, the study has to be replicated at other universities. We consider this as future work.

4.4. Reliability

Reliability is primarily concerned with the question to what extent the study results are dependent on the specific researchers. We addressed the following issues related to reliability in our study design.

By asking specific questions, the moderator of the focus groups could influence the students towards giving the desired answers (researchers' bias). To mitigate this threat, the moderator asked open questions like "What did you do since our last meeting?" during the focus groups. This starts a discussion between the project members without influencing them. Then the moderator asked the participants to go more into detail or to move on to a different topic. A question guide (Mack et al., 2005) had been prepared in advance to make sure that the students gave enough information to answer the research questions. Question guides help the focus group moderator to focus the discussion on relevant topics. If the discussion deviates from the subjects of interest, he can mildly intervene to put it back on track. The question guide used during the focus groups can be found in Appendix A.

An additional potential threat to the reliability of the study results could be that students make imprecise, incomplete, or even non-veridical comments during the focus groups. The following measures were taken to mitigate this risk. First, the focus groups were conducted on a weekly basis to make sure that the students' memories were still present. Second, to verify that the students comments correspond to reality, we used methodological and data-source triangulation (Lethbridge et al., 2005). Apart from establishing a broader view of the research object under study, triangulation allows to verify gained impressions using different data-sources and methods. In particular, we were able to check the students' comments by looking into the minutes of their team meetings and the work artifacts they checked in to the Subversion repositories.

4.5. Ethical issues

The ethical issues resulting from using students in empirical studies were discussed in Section 2.4.2.

5. Related work

Since the late 1980s, researchers have conducted studies to comprehend the design process of software intensive systems (e.g. Curtis et al., 1988; Sonnentag, 1998; Zannier et al., 2007; Brooks, 2009; Tang et al., 2010). The study, presented in this article, is related to this research field. In the following, we outline typical design studies in the field and relate them to our own findings. The presented work covers general design studies, studies of decision making in software projects, and studies on the influence of documentation on the design process. To the best of our knowledge, the influence of architecture decision documentation on the design process has not been empirically investigated so far.

In 1988, Curtis et al. interviewed personnel from 17 large software engineering projects to identify the design activities the teams followed (Curtis et al., 1988). The focus of the study was on how requirements were gathered and how design decisions were made and documented, and how these decisions impacted the design process. They identified three common problems among all analyzed projects: (1) domain knowledge was thinly spread among project members, (2) requirements were often changing or even conflicting, and (3) communication and coordination of tasks was not optimal. Among others, they conclude that staff-wide sharing of (architecture) knowledge has to be facilitated and software development tools should support the representation and management of uncertain design decisions.

Our own findings (almost 25 years later) show that all three identified problems are still perceptible in projects of student software engineers. Using decision viewpoints, however, turned out to at least partially mitigate these problems.

Sonnentag, a German psychologist, analyzed the design process of 40 professional software designers from 16 different software development teams in 1998 (Sonnentag, 1998). After the teams solved a predefined design task, she asked each participant to peer evaluate their team mates. This process was used to distinguish high performers from moderate performers (from the perspective of the team mates). In the subsequent analysis phase, she compared the behavior of the high performers and the moderate performers with respect to problem comprehension, planning, feedback processing,[4] task focus, using visualizations, knowledge of software engineering strategies, and length of experience. Her results include that high performers spent twice as much time on feedback processing than moderate performers. She suggests that high performers, who actively evaluate their design solutions, not only perform better at the present task but also gain more experience for future use in other projects. Surprisingly, she also found that the experience of the participants did not have a significant impact on their behavior regarding the previously mentioned aspects.

Sonnentag emphasizes that the repetitive critical evaluation of design options in the context of requirements (and other forces) helps designers to estimate how far a pursued goal has been achieved. Similarly, our study shows that decision viewpoints, in particular the decision forces viewpoint and the decision relationship viewpoint, provide junior software designers with an organizational structure to support these activities. In another experiment, Tang et al. find that forcing designers to verbalize their design options and reasoning help to bring about better design, especially for junior designers (Tang et al., 2008).

Zannier et al. report on 25 interviews conducted with software designers to develop a model of design decision making in software projects (Zannier et al., 2007). The study focuses on understanding in which situations designers use a rational decision making process versus situations in which the designers follow a naturalistic approach to decision making. Rational decision making, as defined by the authors, is "characterized by the consequential choice of an option among a set of options, with the goal of selecting the optimal solution", whereas naturalistic decision making is "characterized by situation assessment and the evaluation of a single option with a goal of selecting a satisfactory option". The authors found out that designers generally mix both decision making strategies; however, in cases where the design problem was well-defined, the designers under study primarily used rational decision making, whereas the naturalistic approach was preferred to tackle ill-defined problems.

In our own study, we found out that the subjects in the decision view group followed a more rational decision making process than the subjects in the comparison group, although all four projects had a comparably ill-defined design problem in the beginning. This suggests that the documentation of decision views pushed the students towards structuring the design problem better, in order to identify potential solutions and to define criteria (in the decision viewpoint terminology referred to as *forces*) to choose among the solutions. These findings, however, are not contradictory to those of Zannier

[4] Sonnentag defines feedback processing as the comparison of a present situation (here the current version of a software design) with the cognitive representation of the design goal at hand. In other words, feedback processing helps the designer to evaluate how far the design goal has been achieved already.

et al. As conjectured prior to the study, documenting decision views requires designers to think about design options and evaluation criteria upfront, thus they also implicitly require the user of decision views to structure the design problems at hand.

The same conclusions as Zannier et al. were made by Cross. In a review of multiple empirical studies of design activity in different domains (Cross, 2001), Cross acknowledges that designers respond to ill-defined problems by adopting a solution-focused design process. He explains that designers tend to find a *satisfactory* solution rather then systematically generating an *optimal* solution if the problem that needs to be solved is ill-defined. Along with this finding, he states that designers appear to stick to a solution concept as long as possible, even if they encounter shortcomings or difficulties with that solution (this phenomenon is also known as *anchoring* Epley and Gilovich, 2006). In this study, we also found out that the projects in the comparison group searched for design options until they found a satisfactory solution and sticked to these solutions as long as possible. Assuming that designers, as Cross suggests, are by nature solution-focused and subject to anchoring, the creation of decision views helped the students in the decision view group to alleviate the effects of this phenomenon, by forcing them to consider alternatives and explicitly comparing them in the context of the relevant decision forces.

Burge and Brinkman, in an experiment with software engineering students, found that expressing rationale for the choices of algorithms encouraged the students to reflect on their made choices and to actively consider multiple alternatives (Burge and Brinkman, 2010).

The assumption that the documentation of design, and the process of designing itself, mutually interfere with each other has also been examined by Purcell and Gero. Purcell and Gero suggest that the majority of cognitive design activities are too complex for all aspects to be held in short-term memory during the design process (Purcell and Gero, 1998). They advocate that design sketches can serve designers as external memory device, which can be used to reduce the load on working memory. These findings are in line with the statement of the decision view students that the decision views helped them to maintain an overview over decisions made and over all decision forces that had to be taken into consideration.

Similar studies were conducted by Parnas, who, throughout large parts of his academic career, conducted research on documentation and its importance for the software engineering process (Hester et al., 1981; Parnas and Clements, 1986; Parnas, 2009, 2011). Parnas affirms that software design is a decision making process and that documenting software design "forces designers to make decisions and can help them to make better ones". In Parnas and Clements (1986), Parnas and Clements emphasize the importance of designers striving to follow a rational design process. In this work, they stressed the need for documentation to record design decisions, ideally guiding the design process of the development team and serving as a reference during software evolution.

6. Conclusions

Prior to the study, we conjectured that students would use a more rational design process if they use architecture decision views, compared to students who use an ad hoc design approach. We characterize a rational design process using eleven response variables. These eleven response variables were used to analyze the design activities that were carried out by the student project teams.

We have found that in three response variables, the decision relationship viewpoint and the decision forces viewpoint have helped students to follow a more rational design process regarding architectural synthesis and evaluation. Students in this group were better at exploring design options, evaluating the advantages and disadvantages of design options and considering the consequences of combining multiple design options.

On the other hand, the viewpoints were ineffective in helping students in three response areas: to manage requirements, to optimize design regarding complexity, and to explore solution viability by means of prototypes. It appears that something more than the use of viewpoints is needed in order to excel in these three response areas.

We suggest that the identification and documentation of architecture significant requirements and other forces should receive more attention in computer science education. Additionally, we plan to investigate if checklists of typical domain-specific requirements and other forces can at least partially fill the gap regarding requirements documentation. Using prototypes for evaluating design options is an important best-practice we identified in our previous work with professional software architects (van Heesch and Avgeriou, 2011). In our opinion, the use of prototypes should be promoted more in higher computer science education; forces can serve as criteria to evaluate design solutions by means of prototypes. Finally, the current set of decision viewpoints cannot support designers in optimizing a software design with respect to complexity. Additional research is needed to identify metrics for complexity that can be used on a decision level, and to subsequently leverage these metrics by means of decision viewpoints.

Acknowledgement

We would like to thank all participating students from the Software Factory course 2011/2012 at the Fontys University of Applied Science in Venlo, the Netherlands.

Appendix A. Question guide used during the weekly focus groups

The following questions were used as orientation for the moderator of the focus group to make sure that the generally open discussions cover all important aspects of interest. The questions were neither asked verbatim or directly, nor were they necessarily covered in a specific order.

- What has the team done since the last focus group?
- How did they elicit requirements?
- What are the main requirements?
- How do they document requirements?
- Did the team negotiate requirements with the customer?
- How do they prioritize requirements, and which requirements were regarded first and for which reasons?
- Which decisions have been made, and which alternatives were considered?
- How do they make decisions?
- Do the team members challenge each other a lot?
- How does the team lead design discussions?
- Which media, apart from the whiteboard, are used during design discussions?
- What is the team's confidence in the soundness of the decisions? Where are uncertainties?
- Did the team make any assumptions? Which assumptions and why?
- Does the team try to avoid complexity? How?
- Did they make trade-offs between multiple requirements?
- Did they create prototypes, and if so what were they used for?
- How satisfied is the team with the internal process? Do they experience any particular difficulties?

- Note for moderator: Make sure that the team take pictures of all whiteboard sketches

Appendix B. Additional statistics for group assignment

This section presents additional descriptive statistics used for the assignment of project teams to one of the two study groups, i.e. decision view group or comparison group (Figs. B.5–B.8).

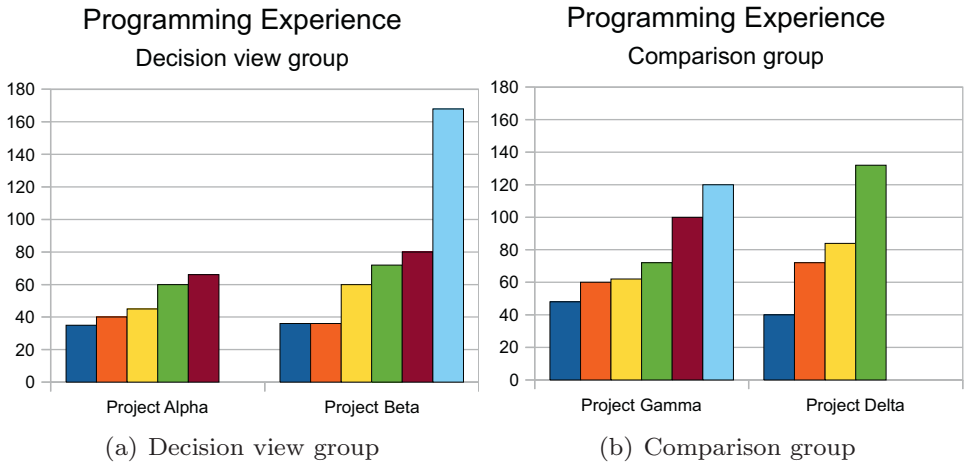
Appendix C. Scrum roles, meetings, and artifacts adopted by the project teams

In this appendix, we describe how far each of the project teams adhered to roles, meetings, and artifacts defined in the Scrum process (CaseVar8). We adopt the definitions of Scrum roles, meetings, and artifacts from Schwaber and Beedle (2002). Table C.7 presents a mapping of the four project teams to the roles, meeting types, and artifacts defined in the Scrum process, which will be described briefly in the following.

Scrum defines different **roles** in the development process. The product owner is a central role in a Scrum-oriented software process. He or she represents the stakeholders' concerns and acts as a spokesman for the customer. The product owner is responsible for ensuring that the team delivers value to the customer's business; furthermore, he or she is primarily responsible for managing the product backlog. Only two of the project teams (projects beta and delta) explicitly assigned the role of the product owner to one of the team members. However, even in these two teams, we did not find conclusive evidence that the roles were adequately filled in. For instance, we observed that all team members edited the product backlogs; furthermore, the product owners did not seem to take over the responsibility for describing or eliciting requirements.

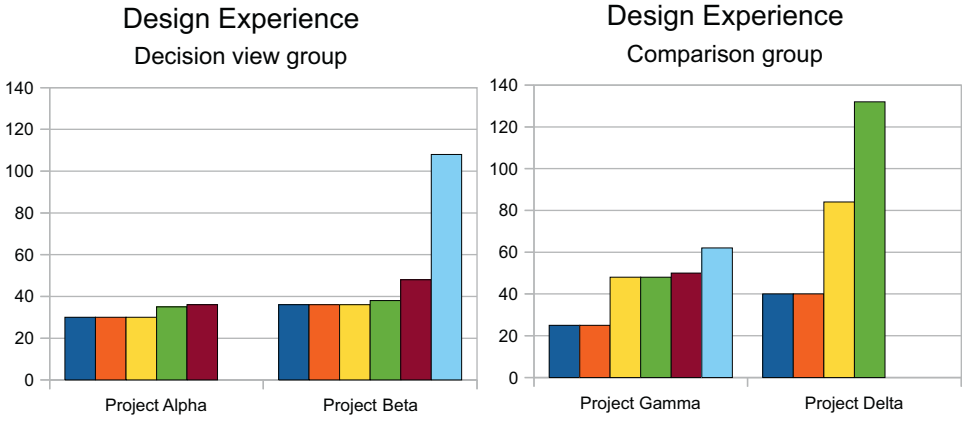
All four project teams explicitly assigned the role of the scrum master to one of the team members. The scrum master is responsible for enabling the team to achieve the project goal and deliverables, e.g. by removing impediments and by making sure that the project team stays motivated and is equipped with an office, and hardware and software needed for the development. Based on our own judgment, these roles were adequately filled in by the respective project team members. In all four project teams, the customer was an external person from a company.

Apart from roles, Scrum defines a number of **meetings**. As Table C.7 shows, the sprint planning meeting is the only meeting that was regularly held by all four project teams. The daily scrum was only held in the first weeks, or not at all (project delta). Time estimation meetings (sometimes referred to as planning poker, Schwaber and Beedle, 2002), were not held by any of the groups. In project gamma, the time estimation was done by the scrum master and communicated to the team. In all other teams, we did not



(a) Decision view group (b) Comparison group

Fig. B.5. Programming experience of the project members in both study groups (CaseVar2). (a) Decision view group and (b) comparison group.



(a) Decision view group (b) Comparison group

Fig. B.6. Software design experience of the project members in both study groups (CaseVar3). (a) Decision view group and (b) comparison group.

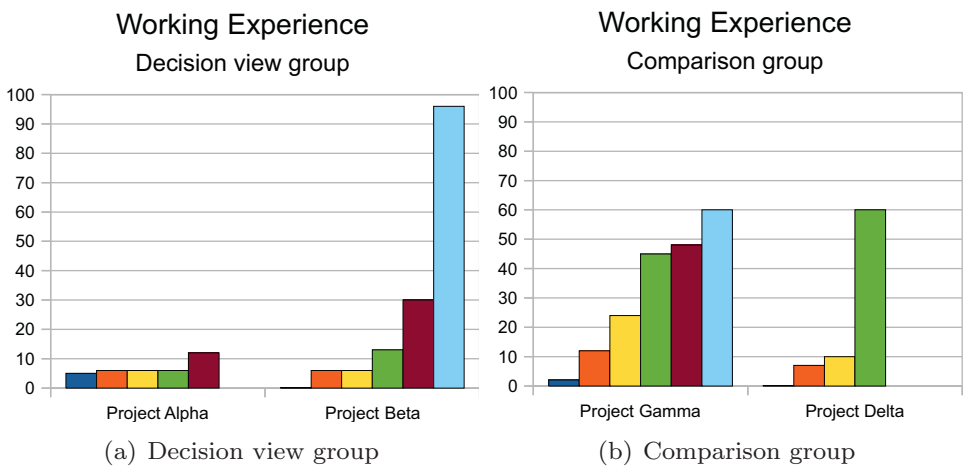


Fig. B.7. Working experience of the project members in both study groups (CaseVar4). (a) Decision view group and (b) comparison group.

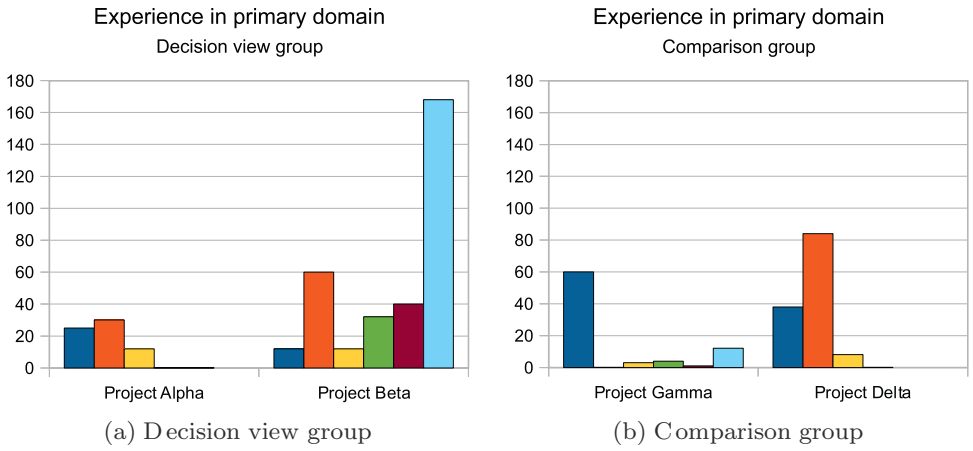


Fig. B.8. Experience in the primary domain of the project members in both study groups (CaseVar7). (a) Decision view group and (b) comparison group.

find any evidence that a systematic time estimation for the backlog items took place. The sprint review and sprint retrospective meetings only took place in projects beta and gamma.

Finally, Scrum defines a number of **artifacts**. Generally, all four project teams had a product backlog and a sprint backlog, and all of them aimed at delivering deployable product increments at the end of each sprint. However, some differences were observed regarding the backlog items. Usually, in Scrum, the product owner defines user stories and other requirements (including non-functional requirements) in the product backlog (Schwaber and Beedle, 2002). In the four project teams, however, we found that the teams primarily stored tasks (e.g. implementation tasks,

Table C.7
Scrum roles, meetings, and artifacts used by the project teams.

	Project alpha	Project beta	Project gamma	Project delta
Roles				
Product owner	Not explicit	Yes	Not explicit	Yes
Team	Yes	Yes	Yes	Yes
Scrum master	Yes	Yes	Yes	Yes
Customer	External	External	External	External
Meetings				
Sprint planning	Explicit meeting	Explicit meeting	Explicit meeting	Explicit meeting
Daily scrum	First week only	First 2 weeks only	First week only	No
Time estimation	No	No	By Scrum master	No
Sprint Review	No	Yes	Yes	No
Sprint retrorespective	No	Yes	Yes	No
Sprint length	3 Weeks	3 Weeks	3 Weeks	2 Weeks
Artifacts				
Product backlog	Yes	Yes	Yes	Yes
Backlog item	Task	Task/user story	Task/user story	Task
Sprint backlog	Yes	Yes	Yes	Yes
Deployable product increment	Yes	Yes	Yes	Yes

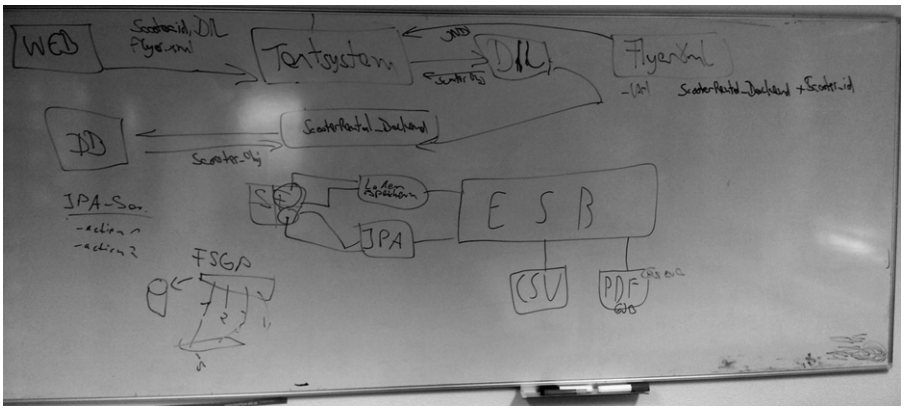


Fig. D.9. Overall architecture envisioned by project team alpha.

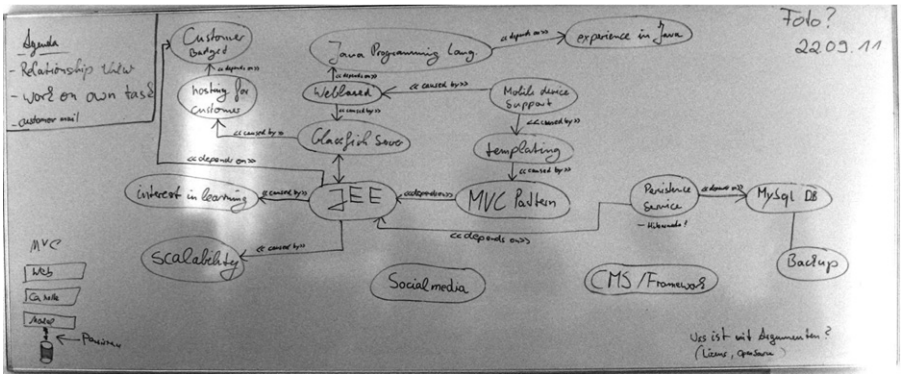


Fig. D.10. Early vision of the architecture created by project team beta (partially using the relationship view.

configuration tasks, review tasks, meetings) in the product backlog. Only teams beta and gamma additionally described a few user stories in the product backlog.

Apart from the Scrum-specific roles and artifacts, each of the teams additionally identified the roles project manager, configuration manager (responsible for setting up required ICT-infrastructure), and quality manager, which were all taken up by different members of the teams. These roles had been suggested by one of the lecturers of the study module.

We conclude that, although Scrum was suggested as development method, none of the four teams rigorously adapted the Scrum process. Only a few Scrum specific instruments were taken over (organization in sprints, scrum master, and backlogs), while other essential parts of Scrum were neglected. Most importantly, the teams did not fill in the role of the product owner correctly.

Appendix D. Initial visions of the architectures

Figs. D.9 and D.10 show examples of early architecture sketches created by the two project teams in the decision view group.

References

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empirical Software Engineering 16 (4), 487–513.

Basili, Victor R., Gianluigi Caldiera, Rombach, Dieter H., 1994. The goal question metric approach. In: Encyclopedia of Software Engineering. John Wiley & Sons, Inc., New York, NY, USA.

Brereton, P., Kitchenham, B., Budgen, D., Li, Z., 2008. Using a protocol template for case study planning. In: Proceedings of the 12th International Conference on Evaluation & Assessment in Software Engineering (EASE 2008).

Brooks, F.P., 2009. The Design of Design: Essays From a Computer Scientist. Addison-Wesley Professional, Boston, MA, USA.

Burge, J.E., Brinkman, B., 2010. Using rationale to assist student cognitive and intellectual development. Human Technology 6 (1), 106–128.

Carver, J.C., Jaccheri, L., Morasca, S., Shull, F., 2010. A checklist for integrating student empirical studies with research and teaching goals. Empirical Software Engineering 15 (1), 35–59.

Corbin, J.M., Strauss, A.L., 2008. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications Inc.

Creswell, J.W., Miller, D.L., 2000. Determining validity in qualitative inquiry. Theory into Practice 39 (3).

Cross, N., 2001. Design cognition: results from protocol and other empirical studies of design activity. Design Knowing and Learning: Cognition in Design Education 7, 9–103.

Curtis, B., Krasner, H., Iscoe, N., 1988. A field study of the software design process for large systems. Communications of the ACM 31 (11), 1268–1287.

Easterbrook, S., Singer, J., Storey, M.A., Damian, D., 2008. Selecting empirical methods for software engineering research. Guide to advanced empirical software engineering 28, 5–311.

Edgewall Software Trac. http://trac.edgewall.org/, February 2012.

Epley, N., Gilovich, T., 2006. The anchoring-and-adjustment heuristic why the adjustments are insufficient. Psychological Science 17 (4), 311–318.

Given, L.M., 2008. The Sage Encyclopedia of Qualitative Research Methods, vol. 1. Sage Publications.

Glaser, B., Strauss, A., 1967. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing Company, Piscataway, NJ, USA.

Glaser, B.G., 1998. Doing Grounded Theory: Issues and Discussions. Sociology Press, Mill Valley, CA.

Harrison, N.B., Avgeriou, P., Zdun, U., 2007. Using patterns to capture architectural decisions. IEEE Software 24 (4), 38–45.

Hester, S.D., Parnas, D.L., Utter, D.F., 1981. Using documentation as a software design medium. Bell System Technical Journal 60 (8), 1941–1977.

Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P., 2007. A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software 80 (1), 106–126.

Host, M., Runeson, P., 2007. Checklists for software engineering case study research. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, pp. 479–481.

ISO Systems and software engineering—architecture description. ISO/IEC/IEEE 42010, September 2011. pp. 1–51.
Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, pp. 109–120.
Jiang, L., Eberlein, A., Far, B.H., Mousavi, M., 2008. A methodology for the selection of requirements engineering techniques. Software and Systems Modeling 7 (3), 303–328.
Kitchenham, B., Pickard, L., Pfleeger, S.L., 1995. Case studies for method and tool evaluation. IEEE Software 12 (4), 52–62.
Kruchten, P., 2004. An ontology of architectural design decisions in software intensive systems. In: Proceedings of the 2nd Groningen Workshop on Software Variability, pp. 54–61.
Lethbridge, T.C., Sim, S.E., Singer, J., 2005. Studying software engineers: data collection techniques for software field studies. Empirical Software Engineering 10 (3), 311–341.
Mack, N., Woodsong, C., MacQueen, K.M., Guest, G., Namey, E., 2005. Qualitative Research Methods: A Data Collector's Field Guide. FLI.
Mannion, M., Keepence, B., 1995. Smart requirements. ACM SIGSOFT Software Engineering Notes 20 (2), 42–47.
Nagappan, N., Maximilien, E.M., Bhat, T., Williams, L., 2008. Realizing quality improvement through test driven development: results and experiences of four industrial teams. Empirical Software Engineering 13 (3), 289–302.
Parnas, D.L., Clements, P.C., 1986. A rational design process: how and why to fake it. IEEE Transactions on Software Engineering 12 (2), 251–257.
Parnas, D.L., 2009. Document based rational software development. Knowledge-Based Systems 22 (3), 132–141.
Parnas, D.L., 2011. Precise documentation: the key to better software. The Future of Software Engineering 12, 5–148.
Patton, M.Q., 2002. Qualitative research and evaluation methods. Sage Publications Inc.
Purcell, A.T., Gero, J.S., 1998. Drawings and the design process: a review of protocol studies in design and other disciplines and related research in cognitive psychology. Design studies 19 (4), 389–430.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14 (2), 131–164.
Schwaber, K., Beedle, M., 2002. Agile Software Development With Scrum. Prentice-Hall, Upper Saddle River, NJ, USA.
Serral, E., Valderas, P., Pelechano, V., 2010. Towards the model driven development of context-aware pervasive systems. Pervasive and Mobile Computing 6 (2), 254–280.
Sjoberg, D.I.K., Dyba, T., Jorgensen, M., 2007. The future of empirical methods in software engineering research. In: Future of Software Engineering, 2007. FOSE'07, pp. 358–378.
Sonnentag, S., 1998. Expertise in professional software design: a process study. Journal of Applied Psychology 3 (5), 703–715.
Strauss, A.L., 1987. Qualitative Analysis for Social Scientists. Cambridge University Press, Cambridge, NY, USA.
Svahnberg, M., Aurum, A., Wohlin, C., 2008. Using students as subjects—an empirical evaluation. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, pp. 288–290.
Tang, A., Tran, M.H., Han, J., Vliet, H.,2008. Design reasoning improves software design quality. In: Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures. Springer-Verlag, pp. 28–42.
Tang, A., Aleti, A., Burge, J., van Vliet, H., 2010. What makes software design effective? Design Studies 31 (6), 614–640, Special Issue Studying Professional Software Design.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Ali Babar, M., 2010. A comparative study of architecture knowledge management tools. Journal of Systems and Software 83 (3), 352–370.
Tigris.org. Subversion. http://subversion.tigris.org, February 2012.
Tyree, J., Akerman, A., 2005. Architecture decisions: demystifying architecture. IEEE Software 22 (2), 19–27.
Urquhart, C., Lehmann, H., Myers, M.D., 2010. Putting the theory back into grounded theory: guidelines for grounded theory studies in information systems. Information Systems journal 20 (4), 357–381.
van Heesch, U., Avgeriou, P., 2011. Mature architecting—a survey about the reasoning process of professional architects. In: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture, IEEE.
van Heesch, U., Avgeriou, P., Hilliard, R., 2012. A documentation framework for architecture decisions. Journal of Systems and Software 85 (4), 795–820.
van Heesch, U., Avgeriou, P., Hilliard, R., 2012. Forces on Architecture Decisions - A Viewpoint. In: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture (WICSA) & 6th European Conference on Software Architecture (ECSA), IEEE.
van der Ven, J.S., Jansen, A.G.J., Nijhuis, J.A.G., Bosch, J., 2006. Design decisions: the bridge between rationale and architecture. In: Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (Eds.), Rationale Management in Software Engineering. Springer-Verlag Heidelberg, pp. 329–348 (Chapter 16).
Verner, J.M., Sampson, J., Tosic, V., Bakar, N.A.A., Kitchenham, B.A., 2009. Guidelines for industrially-based multiple case studies in software engineering. In: Third International Conference on Research Challenges in Information Science. RCIS 2009, IEEE, pp. 313–324.
Wohlin, C., Hoest, M., Runeson, P., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Pub., Boston, NY.
Yin, R.K., 2009. Case Study Research: Design and Methods, Applied Social Research Methods Series, Vol. 5, Fifth ed. Sage Inc.
Zannier, C., Chiasson, M., Maurer, F., 2007. A model of design decision making based on empirical results of interviews with software designers. Information and Software Technology 49 (6), 637–653.

Uwe van Heesch is a senior software engineer at Capgemini Germany. Before joining Capgemini, he worked as a software engineering lecturer at the Fontys University of Applied Sciences in Venlo, the Netherlands. He received his PhD from the University of Groningen, the Netherlands, where he was a member of the Software Engineering research group. His research interests include software architecture, particularly architecture decision modeling, software architectural knowledge management and architecture evaluation. Dr. van Heesch is an active member of the European pattern community and has published several peer-reviewed articles in international journals and conference proceedings.

Dr. Paris Avgeriou is Professor of Software Engineering in the Department of Mathematics and Computing Science, University of Groningen, the Netherlands where he has led the Software Engineering research group since September 2006. Before joining Groningen, he was a post-doctoral Fellow of the European Research Consortium for Informatics and Mathematics (ERCIM). He has participated in a number of national and European research projects directly related to the European industry of Software-intensive systems. He has co-organized several international conferences and workshops (mainly at the International Conference on Software Engineering - ICSE). He sits on the editorial board of Springer Transactions on Pattern Languages of Programming. He has edited special issues in IEEE Software, Elsevier Journal of Systems and Software and Springer Transactions on Pattern Languages of Programming. He has published more than 90 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution and patterns.

Antony Tang is an associate professor in Swinburne University of Technology's Faculty of Information and Computer Technology. His research interests include software architecture design reasoning, software development processes, and knowledge engineering. Tang received his PhD in information technology from the Swinburne University of Technology. He's a member of the ACM.